Desarrollo de un sistema de actualización masiva para la Suite Visión Empresarial (SVE) en clientes en la nube para Pensemos S.A.

Yerson Stewell Ibarra Rueda

Trabajo de grado para optar por el título de Ingeniero de Sistemas

Director

Jose Geralbert Rubiano

Magíster en TIC para la Educación

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingeniería de Sistemas e Informática

Bucaramanga

2026

Dedicatoria

A mi familia, por ser mi mayor apoyo, mi refugio y la fuerza que me impulsó a seguir adelante en cada etapa de este camino. Su amor, paciencia y confianza en mí fueron fundamentales para alcanzar esta meta.

A mi pareja, por su compañía, comprensión y apoyo incondicional durante este proceso. Gracias por estar presente en los momentos de cansancio, motivación e incertidumbre, y por acompañarme con cariño en cada paso.

A mis amigos, por su amistad sincera, por los ánimos en los días difíciles y por recordarme que cada esfuerzo vale la pena cuando se persigue un sueño con constancia.

A mi director de tesis y a mi tutor, por su orientación, dedicación y acompañamiento académico a lo largo de este trabajo. Sus observaciones, consejos y respaldo fueron esenciales para el desarrollo de este proyecto.

Tabla de contenido

Págs.

Lista de Tablas

Págs.

Lista de Figuras

Págs.

Lista de Apéndices

Los apéndices están adjuntos y pueden visualizarse en la base de datos de la biblioteca UIS.

Apéndice A. Evidencia de acceso al sitio de validación final en entorno de Pensemos

Apéndice B. Logs de Jenkins y evidencias del despliegue

Glosario

Anexo: sección complementaria ubicada al final de la tesis donde se incluye material de apoyo como evidencias, capturas, logs o archivos de respaldo.

Apache Ant: herramienta de automatización de compilación y ejecución de tareas que permite orquestar procesos repetitivos en proyectos de software.

Backup: copia de seguridad de archivos o bases de datos destinada a recuperar información en caso de error o pérdida.

CI/CD: práctica de integración continua y entrega o despliegue continuo que automatiza compilación, pruebas y publicación de software.

DevOps: enfoque cultural y técnico que integra desarrollo y operaciones mediante colaboración, automatización y retroalimentación continua.

Jenkins: servidor de automatización de código abierto usado para orquestar pipelines de integración y despliegue continuo.

Log: registro generado por un sistema o aplicación donde se documentan eventos, errores, tiempos y resultados de ejecución.

Pipeline: secuencia automatizada de pasos que ejecuta tareas de construcción, validación y despliegue de software.

Plugin: componente adicional que amplía las funcionalidades de una herramienta, como Jenkins, para adaptarla a necesidades específicas.

Repositorio SVN: almacén central de código fuente y archivos del proyecto administrado con Subversion.

Subversion (SVN): sistema de control de versiones utilizado para gestionar cambios en archivos y código fuente de un proyecto.

Tomcat: servidor web y contenedor de aplicaciones Java usado para desplegar archivos WAR.

VEAntTask: conjunto de tareas personalizadas de Ant desarrollado para apoyar la compilación, empaquetado y automatización del sistema SVE.

VEFile: estructura de archivos y configuraciones del entorno de despliegue de la Suite Visión Empresarial.

WAR: archivo empaquetado de una aplicación web Java que se despliega en servidores como Tomcat.

Resumen

Título: Desarrollo de un sistema de actualización masiva para la Suite Visión Empresarial (SVE) en clientes en la nube para Pensemos S.A.*

Autor: Yerson Stewell Ibarra Rueda**

Palabras clave: Suite Visión Empresarial, actualización masiva, CI/CD, DevOps, Jenkins, Subversion, automatización, despliegue en la nube.

Descripción: Este trabajo de grado presenta el diseño e implementación de un sistema para automatizar la actualización masiva de la Suite Visión Empresarial (SVE) en clientes alojados en la nube. El proceso manual y secuencial de actualización de la plataforma para más de 60 clientes corporativos y gubernamentales generaba altos tiempos de ejecución, dependencia operativa, cuellos de botella y riesgo de errores, lo que motivó el desarrollo de una solución basada en principios de CI/CD y las metodologías DevOps.

La arquitectura tecnológica desarrollada integra Jenkins como orquestador principal, junto con Subversion, Apache Ant, VEAntTask, servidores Apache Tomcat y bases de datos Oracle. Esta integración permite ejecutar de forma controlada y simultánea las etapas de obtención de versiones, compilación, empaquetado de archivos WAR, despliegue y actualización de esquemas por entorno. Adicionalmente, se incorporaron mecanismos de validación humana en puntos críticos, trazabilidad, respaldo y recuperación automatizada para mitigar el riesgo operativo, resguardar la información y mejorar la confiabilidad del proceso.

La solución fue validada con éxito en entornos de prueba proporcionados por Pensemos S.A., evidenciando que la automatización disminuye drásticamente la intervención manual, optimiza los tiempos de despliegue, mitiga los fallos técnicos y mejora de manera sustancial la eficiencia, escalabilidad y seguridad de todo el ciclo de actualización de software empresarial moderno.

---

* Trabajo de Grado

** Facultad de Ingenierías Fisicomecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: Jose Geralbert Rubiano. Magíster en TIC para la Educación.

Abstract

Title: Development of a mass update system for the Suite Visión Empresarial (SVE) in cloud-based clients for Pensemos S.A.*

Author: Yerson Stewell Ibarra Rueda**

Keywords: Suite Visión Empresarial, mass update, CI/CD, DevOps, Jenkins, Subversion, automation, cloud deployment.

Description: This undergraduate thesis presents the design and implementation of a system to automate the mass update process of the Suite Visión Empresarial (SVE) for cloud-hosted clients. The manual and sequential update process of the platform across more than 60 corporate and government clients involved long execution times, operational dependency, bottlenecks, and a high risk of errors, which motivated the proposal of a solution based on CI/CD principles and DevOps methodologies.

The developed technical architecture integrates Jenkins as the main orchestrator, together with Subversion, Apache Ant, VEAntTask, Apache Tomcat servers, and Oracle databases. This integration enables the controlled and simultaneous execution of version retrieval, compilation, WAR file packaging, deployment, and database schema update stages per environment. In addition, human validation mechanisms at critical checkpoints, traceability, backup, and automated recovery features were incorporated to mitigate operational risk, safeguard information, and improve overall process reliability.

The solution was successfully validated in test environments provided by Pensemos S.A., proving that automation drastically reduces manual intervention, optimizes deployment times, mitigates technical failures, and substantially improves the efficiency, scalability, and security of the entire update cycle for modern enterprise software.

---

* Thesis/Final Degree Project

** Faculty of Physical-Mechanical Engineering. School of Systems and Computer Engineering. Advisor: Jose Geralbert Rubiano. Master in ICT for Education.

Introducción

En la actualidad, la gestión eficiente de actualizaciones de software en entornos empresariales distribuidos es un reto fundamental para garantizar la estabilidad, seguridad y continuidad operativa de las organizaciones. La Suite Visión Empresarial (SVE), desarrollada por Pensemos S.A., es una plataforma modular utilizada por múltiples clientes en la nube para administrar procesos críticos que afectan desde la gestión estratégica hasta la operativa y documental. Debido a la arquitectura distribuida y la creciente base de usuarios, el proceso de actualización manual, secuencial y dependiente del equipo de soporte se vuelve un cuello de botella operativo que limita la escalabilidad y aumenta la posibilidad de errores y tiempos de inactividad.

El avance de las metodologías DevOps y las prácticas de integración y entrega continua (CI/CD) ofrecen un marco teórico y tecnológico para transformar estos procesos a través de la automatización, la orquestación y la colaboración estrecha entre desarrollo y operaciones. Sin embargo, la implementación de estas prácticas en sistemas complejos y modulares como la SVE plantea retos específicos relacionados con la coordinación masiva de despliegues, el control de versiones, la validación humana en etapas críticas y la integración en entornos cloud heterogéneos.

Este trabajo de grado propone el diseño e implementación de un sistema basado en pipelines CI/CD y principios DevOps para gestionar y ejecutar de forma masiva y controlada las actualizaciones de las instalaciones de SVE alojadas en la nube. Con ello se busca optimizar la administración, reducir la carga operativa del equipo de soporte, disminuir riesgos técnicos y operativos, y acelerar el ciclo de entrega continua, alineando el proceso con las mejores prácticas internacionales para software empresarial moderno.

Los resultados esperados incluirán un sistema que facilitará actualizaciones seguras, con control y supervisión humana en puntos clave para asegurar la calidad y confiabilidad del sistema. Asimismo, se generará documentación técnica y manuales que permitirán la transferencia y adopción efectiva de la solución en el contexto operativo de Pensemos S.A., contribuyendo al fortalecimiento tecnológico y la competitividad de la organización.

1 Planteamiento y justificación del problema

La Suite Visión Empresarial (SVE) es una solución software modular desarrollada por Pensemos S.A., diseñada para integrar herramientas clave de gestión en una sola plataforma web que facilita la administración estratégica, operativa y documental de organizaciones públicas y privadas. La SVE permite centralizar la información, apoyar la toma de decisiones, promover el control y seguimiento de objetivos, e impulsar la mejora continua a través de módulos especializados como indicadores, documentos, riesgos, planes, entre otros. Actualmente, está en uso por más de 60 clientes corporativos y entidades gubernamentales que operan sus instancias en infraestructura cloud.

Sin embargo, el proceso de actualización de estas instalaciones basadas en la nube se ejecuta de manera manual y secuencial, dependiendo del equipo de soporte para actualizar cliente por cliente mediante herramientas heterogéneas y procedimientos manuales, lo que genera riesgos operativos altos y limita la escalabilidad del sistema. La intervención humana intensiva incrementa la posibilidad de errores y omisiones, dificulta el control de versiones y alarga los tiempos de despliegue, dificultando la respuesta efectiva ante la evolución rápida de necesidades y la creciente base de usuarios.

El sistema a desarrollar automatizará de forma masiva principalmente dos tipos de actualizaciones críticas para la Suite Visión Empresarial: las actualizaciones de bases de datos y las actualizaciones de despliegues de aplicaciones empaquetadas en archivos WAR dentro de servidores Apache Tomcat.

Ante esta problemática surge la pregunta de investigación: ¿Es factible desarrollar un sistema automatizado basado en pipelines de integración y despliegue continuo (CI/CD) con un enfoque DevOps que permite orquestar y ejecutar las actualizaciones masivas de SVE en la nube, mejorando la eficiencia, reduciendo riesgos técnicos y facilitando la gestión operativa con validación humana en puntos críticos?

A nivel internacional, DevOps y CI/CD se han consolidado como estándares fundamentales para lograr entregas frecuentes, confiables y seguras de software en entornos cloud, permitiendo optimizar la colaboración entre equipos de desarrollo y operación y disminuyendo tiempos y errores en despliegues (Red Hat, 2022). En Colombia y la región, aunque la transformación digital avanza, todavía existen retos en la implementación de estas metodologías en sistemas empresariales complejos con múltiples clientes, lo que hace necesaria una adaptación enfocada a contextos específicos (URepublicana, 2024).

2 Objetivos

2.1 Objetivo general

Desarrollar un sistema que permita gestionar y ejecutar de forma masiva y controlada la actualización de las instalaciones de la Suite Visión Empresarial (SVE) alojadas en la nube, utilizando herramientas de CI/CD con enfoque DevOps para mejorar la administración y reducir la carga operativa del equipo de soporte de PENSEMOS S.A.

2.2 Objetivos específicos

- Analizar los sistemas y herramientas actuales orientadas a la gestión y ejecución de actualizaciones masivas, evaluando su arquitectura, funcionalidades, ventajas y limitaciones para definir los requisitos funcionales y no funcionales, así como la mejor estrategia tecnológica que se aplicará en el desarrollo del sistema.
- Diseñar el sistema de actualización masiva automatizada para la Suite Visión Empresarial en la nube, definiendo la arquitectura, componentes y flujos de integración de pipelines CI/CD.
- Desarrollar el sistema de actualización masiva automatizada, implementando los pipelines CI/CD diseñados e integrando los componentes necesarios para la ejecución efectiva y controlada de las actualizaciones en la nube de SVE.
- Validar el sistema de actualización masiva en un ambiente de prueba controlado, garantizando que cumpla con los requisitos funcionales y no funcionales definidos, asegurando su desempeño, seguridad y confiabilidad.

- Generar la documentación técnica completa del sistema desarrollado, incluyendo manuales de usuario y procedimientos operativos, con el fin de facilitar su implementación, mantenimiento y transferencia al equipo de soporte de Pensemos S.A.

3 Marco de referencia

3.1 Marco teórico

3.1.1 Fundamentos de la integración continua y entrega continua (CI/CD)

La integración continua (CI) es una práctica fundamental que consiste en fusionar de manera frecuente y automática los cambios de código realizados por diversos desarrolladores en un repositorio compartido. Esta práctica permite realizar compilaciones y pruebas automatizadas constantes para detectar errores rápidamente, garantizando que el software se mantenga en estado desplegable (Red Hat, 2022). La entrega continua (CD), por su parte, automatiza la entrega de versiones validadas hacia entornos de producción o preproducción, facilitando que cada cambio probado pueda ser desplegado con mínima intervención manual (ServiceNow, 2025).

Figura 1

Esquema del flujo de integración y distribución continuas (CI/CD)

Nota. Tomado de La integración y la distribución continuas (CI/CD)

Las fases principales de CI/CD incluyen la integración del código, la automatización de la construcción, la ejecución de pruebas, y finalmente la entrega o despliegue continuo. Entre sus

beneficios destacan la reducción de errores, la mejora de la calidad del software y la aceleración en las entregas. Sin embargo, también existen desafíos como la complejidad en la integración de herramientas y la necesidad de cambio cultural para su adopción efectiva (Atlassian, s.f.).

3.1.2 Principios y características del enfoque DevOps

DevOps es un paradigma que busca cerrar la brecha entre los equipos de desarrollo y operaciones para lograr una colaboración continua y efectiva, apoyada en la automatización y el monitoreo constante (Red Hat, 2022). Sus principios fundamentales incluyen la colaboración transversal, la entrega rápida y confiable, la automatización de procesos y la monitorización de los sistemas en producción (ServiceNow, 2025).

Esta cultura organizacional se basa en pipelines CI/CD, gestión de infraestructura como código, pruebas automatizadas y despliegues automáticos, buscando reducir tiempos de respuesta y mejorar la calidad del software entregado.

3.1.3 Relación entre CI/CD, DevOps y la mejora de actualizaciones

La combinación de prácticas CI/CD y DevOps permite a las empresas optimizar el proceso de actualizaciones de software, logrando despliegues frecuentes, más seguros y controlados. En entornos complejos y modulares como el de la Suite Visión Empresarial, estas metodologías facilitan la gestión coordinada de múltiples instancias, asegurando trazabilidad, calidad y capacidad de adaptación rápida (Platform Engineering, 2025).

Este enfoque no solo reduce los riesgos asociados a las actualizaciones manuales, sino que también mejora la eficiencia operativa y la capacidad de respuesta ante incidentes, alineando los procesos con los estándares internacionales de software empresarial en la nube.

3.2 Estado del arte

3.2.1 Ansible Automation Platform (AAP)

Ansible Automation Platform (AAP) de Red Hat es una plataforma integral de automatización que facilita a las organizaciones la creación, ejecución y gestión de automatizaciones de IT a escala empresarial. AAP permite a arquitectos, desarrolladores y equipos operativos definir, delegar y controlar tareas automatizadas de manera centralizada, incluyendo la instalación y actualización masiva de software en infraestructuras distribuidas, ya sea en entornos on-premises o cloud (Red Hat, 2022).

Entre las funcionalidades clave que soportan la gestión de actualizaciones masivas están los playbooks de Ansible, que permiten definir flujos repetibles y reproducibles para la instalación y actualización de software bajo un formato declarativo; el controlador de automatización que facilita la orquestación y programación de tareas en múltiples nodos concurrentemente; y las colecciones certificadas que aportan módulos validados para la configuración y despliegue en distintas plataformas y servicios. Además, AAP proporciona dashboards e informes en tiempo real para medir el impacto, éxito y ROI de las automatizaciones implementadas (Red Hat, 2025).

La plataforma también incorpora capacidades avanzadas como la automatización basada en eventos (event-driven automation), integración con sistemas de gestión de secretos para seguridad, y soporte para entornos híbridos y multi-cloud. Estas características hacen que Ansible

Automation Platform sea una solución robusta y escalable para la gestión automatizada de actualizaciones en sistemas modulares empresariales, reduciendo tiempos, errores operativos y riesgos relacionados con los despliegues manuales (Red Hat, 2025).

3.2.2 Azure Update Management Center

Azure Update Management Center es un servicio unificado que ayuda a las organizaciones a administrar y gobernar las actualizaciones de software para máquinas que ejecutan sistemas operativos Windows y Linux, tanto en entornos Azure, on-premises como en otras nubes mediante Azure Arc (Microsoft, 2025). Este servicio proporciona un panel centralizado para controlar el cumplimiento de actualizaciones, permitiendo programar instalaciones en ventanas de mantenimiento definidas o realizar actualizaciones en tiempo real, optimizando la seguridad y disponibilidad de la infraestructura.

Entre sus características destacadas se encuentran el agrupamiento dinámico de máquinas basado en criterios definidos, opciones flexibles de aplicación de parches como parches automáticos en máquinas virtuales, mantenimiento programado y hasta la aplicación de hotpatches para minimizar reinicios y tiempos de inactividad (Microsoft, 2025). Además, ofrece capacidades de monitoreo detallado, reportes personalizados y alertas configurables para asegurar la trazabilidad y control exhaustivo del proceso de actualización.

La integración nativa con Azure y Azure Arc facilita la gestión de infraestructuras híbridas y multi-cloud, con control granular de acceso por roles (RBAC), garantizando la seguridad y cumplimiento de políticas de actualización en grandes organizaciones (Microsoft, 2025). Gracias a estas funcionalidades, Azure Update Management Center es una solución poderosa para la

automatización y orquestación de actualizaciones masivas, alineándose con las mejores prácticas de operación modernas.

4 Metodología

El proyecto de actualización masiva de la Suite Visión Empresarial (SVE) en la nube implica un entorno dinámico y complejo con múltiples clientes y componentes modulares. La metodología incremental permite integrar cambios y nuevas funcionalidades de manera ágil durante el desarrollo, facilitando la adaptación a requisitos emergentes o ajustes técnicos sin afectar el avance global.

4.1 Metodología incremental

La metodología incremental es un enfoque de desarrollo de software en el que el proyecto se divide en partes pequeñas y manejables llamadas incrementos. Cada incremento se planifica, diseña, desarrolla y prueba de manera independiente para entregar un producto funcional que aporta valor desde etapas tempranas.

Este enfoque se basa en ciclos iterativos, donde cada iteración añade nuevas funcionalidades o mejoras a las ya existentes, permitiendo una evolución progresiva del sistema. A diferencia de los métodos tradicionales que entregan el producto completo al final, la metodología incremental proporciona entregas parciales y funcionales durante todo el ciclo de desarrollo, lo que facilita la validación temprana, la retroalimentación continua y la adaptación a cambios de requisitos. Además, reduce riesgos y costos asociados a cambios tardíos, al permitir ajustar y mejorar el producto en cada incremento.

Este modelo es especialmente útil en contextos dinámicos y con requisitos cambiantes, como proyectos de integración continua y despliegue continuo (CI/CD) en entornos DevOps, ya que asegura entregas frecuentes, funcionales y de alta calidad que alinean el desarrollo con las necesidades reales del cliente.

4.2 Utilidad metodología incremental en el sistema

La metodología incremental es particularmente útil para este proyecto de actualización masiva en la Suite Visión Empresarial porque permite abordar la complejidad del sistema de manera progresiva y controlada. Al dividir el proyecto en incrementos funcionales, se facilita la gestión de las diferentes partes del sistema alojado en la nube, asegurando entregas parciales que pueden ser evaluadas y validadas a tiempo. Esto reduce significativamente los riesgos técnicos asociados a actualizaciones manuales y secuenciales, permitiendo detectar y corregir errores temprano.

Además, la metodología fomenta la colaboración continua entre desarrollo y operaciones, esencial para integrar prácticas CI/CD y DevOps que son núcleo del proyecto. De esta forma, se mejora la eficiencia operativa, se disminuye la carga de trabajo del equipo de soporte, Página 7 de 15 y se acelera la frecuencia y calidad de las actualizaciones, alineándose con las necesidades de escalabilidad y fiabilidad de Pensemos S.A.

4.3 Comunicación y Seguimiento

Para asegurar una comunicación fluida y un seguimiento efectivo del proyecto, se utilizarán herramientas digitales que permitan la colaboración en tiempo real y la gestión transparente de las tareas y horarios. Trello será la plataforma principal para la gestión ágil del proyecto, facilitando

la organización de tareas en tableros visuales mediante tarjetas que representan actividades, con estados claros y asignación de responsables. Esto permitirá mantener una visión actualizada del progreso, prioridades y bloqueos de cada ciclo de trabajo.

Adicionalmente, se implementará Jibble para el control de tiempos y seguimiento de la asistencia del equipo, lo cual es vital para medir dedicación y productividad, especialmente en un entorno de trabajo remoto o distribuido. La combinación de estas herramientas contribuye a fomentar la responsabilidad, la transparencia y la colaboración continua entre miembros del equipo, alineándose con los principios ágiles de comunicación constante y mejora continua.

4.4 Fases y Actividades del Proyecto

El desarrollo de este proyecto se estructura en fases que organizan las actividades en etapas secuenciales. Cada fase cuenta con actividades definidas que permiten avanzar progresivamente desde el diagnóstico inicial hasta el despliegue y validación final del sistema automatizado para actualización masiva de la Suite Visión Empresarial.

Esta división facilita la planificación, seguimiento y control del proyecto, asegurando que se cumplan los objetivos en tiempos definidos y con la calidad esperada.

Fase 1: Análisis y Planificación

Aquí se recopilan y analizan los requisitos y se establecen los lineamientos técnicos.

- Análisis de requisitos y planificación del proyecto
- Recopilación y definición de requisitos

●Análisis comparativo de sistemas y herramientas

● Diseño preliminar del sistema

● Revisión de arquitectura y herramientas CI/CD para la solución

Fase 2: Desarrollo Iterativo

Fase de ejecución práctica del desarrollo del sistema mediante ciclos iterativos.

● Desarrollo y codificación por incrementos funcionales

● Gestión y seguimiento por incrementos

● Revisión y ajuste periódico del flujo de trabajo para optimizar eficiencia

Se obtendrá la implementación progresiva de incrementos funcionales plenamente operativos, con pruebas parciales y validaciones internas que garanticen la calidad en cada entrega. Habrá un control de la gestión y seguimiento continuo para optimizar el flujo de trabajo y responder rápidamente a desviaciones o dificultades. Además, se consolidará la integración con herramientas de CI/CD y la colaboración DevOps entre equipos, mostrando evolución constante y entregas parciales que agregan valor funcional al sistema.

Fase 3: Retroalimentación y Ajuste

● Revisión y evaluación formal de cada incremento

● Recopilación de retroalimentación del equipo

● Actualización de la planificación y prioridades para los siguientes incrementos

Se obtendrá una evaluación formal, estructurada y documentada de cada incremento funcional mediante feedback del equipo de desarrollo. Se realizarán ajustes en requisitos, prioridades y diseños basados en esta información, mejorando la alineación con las necesidades reales y corrigiendo desviaciones tempranas. Esta fase asegura que el proyecto se mantenga flexible y adaptable, facilitando una evolución guiada por la experiencia e información obtenida.

Fase 4: Documentación, Pruebas y Despliegue

Se finaliza el proyecto mediante validaciones, documentación y despliegue piloto.

- Documentación técnica y manuales de usuario
- Pruebas de aceptación y validación final
- Despliegue piloto y monitoreo inicial

Se completará la documentación técnica detallada que incluye manuales de usuario, procedimientos operativos. Además. Se ejecutará un despliegue piloto controlado con monitoreo que permita validar la operación en ambientes simulados.

5 Diseño preliminar del sistema

Como base del diseño se consideran las herramientas principales del ecosistema actual: un orquestador CI/CD para ejecutar el flujo y centralizar resultados, Apache Ant para modelar el flujo técnico por fases, Java y VEAntTask para tareas personalizadas, SVN como fuente de versionado, VeUpdateSuite para la actualización de componentes y base de datos, y Tomcat como plataforma de despliegue. Esta combinación se propone porque permite integrar en un solo flujo la

construcción del software, la actualización de datos y la puesta en marcha del entorno objetivo. Para tener en cuenta en el diseño, se priorizan los siguientes criterios prácticos:

1. Definir responsabilidades claras por herramienta para evitar acoplamientos innecesarios.

2. Separar fases críticas (checkout, build, update BD y deploy) para facilitar diagnóstico.

3. Incluir validaciones tempranas de propiedades y conectividad antes de ejecutar cambios.

4. Mantener evidencia de tiempos y versiones para asegurar trazabilidad del proceso.

El diseño previo propone un flujo único y ordenado que inicie en un orquestador CI/CD y termine con el sitio actualizado en Tomcat. Como punto de partida, se plantea que el pipeline ejecute una secuencia estándar: inicialización de variables y versiones, checkout de código y herramientas desde SVN, compilación y empaquetado de artefactos, actualización de base de datos (Oracle o SQL Server) y despliegue final. Este diseño se enfoca en tres principios funcionales:

1. Estandarización del proceso para que todas las ejecuciones sigan la misma ruta técnica.

2. Control por fases para identificar con precisión en qué etapa falla una ejecución.

3. Trazabilidad completa de versiones, tiempos y resultado de cada paso.

Con este enfoque, el diseño previo establece que la automatización no solo construya y despliegue, sino que también genere evidencia técnica suficiente para auditoria, soporte y mejora continua.

5.1 Arquitectura propuesta

La arquitectura se define como la materialización técnica del diseño previo y organiza responsabilidades por componentes conectados entre sí. El orquestador CI/CD actuará como coordinador del proceso; Ant ejecutará la lógica del pipeline; Java y VEAntTask habilitaran tareas especializadas; SVN suministrará código y metadatos de version; VeUpdateSuite y scripts SQL gestionan actualizaciones de base de datos; y Tomcat recibirá las operaciones de undeploy, reinicio y deploy del artefacto final.

La relación entre ambos apartados es directa: el diseño previo define que debe ocurrir y en que orden, mientras la arquitectura propuesta define con que componentes y mecanismos se cumplirá ese objetivo. En consecuencia, la arquitectura garantiza que cada fase del diseño tenga un responsable técnico explícito y una integración controlada dentro del pipeline.

Como resultado esperado de esta relación diseño-arquitectura, el pipeline deberá ofrecer una ejecución repetible, observable y alineada con la consistencia aplicación-BD, minimizando riesgos operativos durante la actualización de entornos de prueba.

Nota: La descripción presentada corresponde a un nivel preliminar. Para consulta de detalle técnico, la documentación técnica del proyecto desarrollará de forma ampliada el flujo completo por etapas, la secuencia de ejecución por paso y la especificación detallada del diseño final implementado. Esta base también orientará la definición y trazabilidad de los requisitos funcionales y no funcionales que se desarrollarán en fases posteriores.

6 Análisis y Planificación

6.1 Análisis de requisitos y planificación del proyecto

El proyecto inició con la identificación de una necesidad operativa en Pensemos S.A., relacionada con la ejecución manual y secuencial de actualizaciones sobre instancias cloud de la Suite Visión Empresarial SVE. Este escenario evidenció la necesidad de diseñar un mecanismo automatizado que permitiera ejecutar actualizaciones de bases de datos y despliegues de aplicación de manera controlada.

Con base en este diagnóstico, se definieron los requisitos funcionales y no funcionales del sistema, estableciendo las capacidades que debía ofrecer, como gestión de clientes y entornos, ejecución automatizada de scripts, respaldo previo a actualizaciones, despliegue de artefactos WAR y generación de trazabilidad por ejecución. De igual manera, se identificaron atributos de calidad como escalabilidad, mantenibilidad e integración con la infraestructura existente.

La planificación del proyecto se orientó a partir de la revisión de herramientas y tecnologías disponibles, priorizando aquellas que se ajustaran al ecosistema tecnológico de la organización y permitieran una implementación incremental con bajo riesgo de adopción. Esto sirvió como base para estructurar la arquitectura inicial y preparar el desarrollo iterativo de la solución.

6.2 Recopilación y definición de requisitos

Con base en el análisis del problema, los objetivos del proyecto y el contexto operativo de la Suite Visión Empresarial SVE, se realizó la recopilación y definición de los requisitos necesarios para orientar el desarrollo de la solución. Este proceso permitió identificar de manera clara las

funcionalidades que debía cumplir el sistema, así como las condiciones de calidad y restricciones técnicas que debían considerarse para garantizar su correcta implementación.

Los requisitos formulados responden a la necesidad de automatizar de forma masiva y controlada las actualizaciones de clientes en la nube, asegurando trazabilidad, confiabilidad, seguridad y compatibilidad con la infraestructura existente de Pensemos S.A. En este sentido, los requisitos funcionales establecen qué debe hacer el sistema, mientras que los no funcionales definen cómo debe comportarse para ser útil, seguro y sostenible en el entorno de operación.

6.2.1 Requisitos funcionales

Tabla 1

Requisitos funcionales

ID	Descripción	Prioridad	Criterio de aceptación
RF1	El sistema gestiona un catálogo de clientes y entornos de SVE en la nube, almacenando datos de conexión a servidores, bases de datos y ambientes para orquestar actualizaciones masivas centralizadas.	Alta	Para el piloto se registran correctamente datos asociados a la nube del cliente y se usan en una ejecución real de actualización.
RF2	El sistema permite seleccionar uno o varios clientes y entornos por ejecución, que actualizará base de datos y aplicación (WAR/Tomcat).	Alta	En pruebas de validación se ejecutan actualizaciones de BD y WAR combinadas, evidenciando que el alcance configurado se respeta en los logs y resultados del pipeline.

RF3	El sistema ejecuta de forma automatizada los scripts de actualización de base de datos sobre las instancias seleccionadas, respetando el orden definido y usando credenciales parametrizadas.	Alta	En ambiente de prueba los scripts de BD se aplican automáticamente a los entornos definidos, dejando registros por cliente y entorno sin intervención manual en la ejecución.
RF4	El sistema despliega de forma automatizada los artefactos WAR de la SVE en los servidores Apache Tomcat configurados, siguiendo el flujo de parada, despliegue y arranque por entorno.	Alta	En el piloto se despliega una nueva versión de la SVE en los Tomcat de al menos 2 entornos, con evidencia de stop/deploy/start exitosos en los logs del pipeline.
RF5	El sistema realiza un backup de la base de datos de cada cliente/entorno antes de aplicar actualizaciones que la afecten, registrando la ubicación de los respaldos.	Alta	Antes de una actualización de BD se genera un backup y se verifica al menos una restauración exitosa en ambiente de prueba a partir de dicho respaldo.
RF6	El sistema orquesta un pipeline CICD que integra checkout desde SVN, compilación/empaquetado con VEAntTask y despliegue multi-entorno, con pasos definidos.	Alta	Se ejecuta el pipeline completo mostrando en los logs cada etapa y su resultado, sin necesidad de pasos manuales intermedios fuera del flujo definido.
RF7	El sistema registra logs detallados por cliente y entorno para cada etapa del pipeline, incluyendo tiempos, errores y estado	Alta	Para una ejecución de prueba se dispone de logs estructurados donde se identifica por cliente/entorno el estado de cada paso

	final (éxito/fallo) de la actualización.		y el tiempo consumido.
RF8	El sistema proporciona retroalimentación en consola al equipo de soporte sobre el resultado de las actualizaciones, resaltando entornos con errores y pasos donde falló la ejecución.	Media	Al terminar una ejecución masiva se genera un resumen visible con la lista de entornos exitosos y fallidos y el paso donde ocurrió el error.
RF9	El sistema parametriza su comportamiento a través de archivos de propiedades (módulos, credenciales, rutas, cantidad de entornos, prefijos), validando que las propiedades obligatorias estén definidas.	Alta	En pruebas, cuando falta una propiedad obligatoria por entorno, el pipeline se detiene con mensaje de error específico; cuando están completas, la ejecución avanza sin interrupciones.
RF10	El sistema soporta la generación y mantenimiento de documentación técnica y procedimientos operativos que describen el pipeline, prerrequisitos y manejo de errores.	Media	Se dispone de documentación actualizada usada efectivamente por el equipo de soporte para ejecutar el pipeline y resolver incidencias comunes.

6.2.2 Requisitos no funcionales

Tabla 2

Requisitos no funcionales.

ID	Descripción	Prioridad	Criterio de aceptación
RF1	El sistema soporta un aumento progresivo en la cantidad de clientes y entornos a actualizar sin degradación severa	Media	Para el piloto se registran correctamente datos asociados a la nube del cliente y se usan

	del rendimiento, mediante ejecuciones masivas planificadas.		en una ejecución real de actualización.
RF2	El sistema minimiza el riesgo de dejar instalaciones inconsistentes gracias a backups previos, validaciones de prerrequisitos y control de errores en cada paso del pipeline.	Alta	En pruebas de validación se ejecutan actualizaciones de BD y WAR combinadas, evidenciando que el alcance configurado se respeta en los logs y resultados del pipeline.
RF3	El sistema protege credenciales de SVN, bases de datos y servidores Tomcat, evitando su exposición en texto plano en logs y reduciendo accesos no autorizados.	Alta	En ambiente de prueba los scripts de BD se aplican automáticamente a los entornos definidos, dejando registros por cliente y entorno sin intervención manual en la ejecución.
RF4	El sistema permite planificar ejecuciones en ventanas de mantenimiento, minimizando el tiempo de indisponibilidad para los usuarios finales durante las actualizaciones.	Media	En el piloto se despliega una nueva versión de la SVE en los Tomcat de al menos 2 entornos, con evidencia de stop/deploy/start exitosos en los logs del pipeline.
RF5	El sistema permite rastrear qué versión de SVE y qué scripts de base de datos se aplicaron a cada cliente y entorno, con fecha y pipeline asociado.	Alta	Antes de una actualización de BD se genera un backup y se verifica al menos una restauración exitosa en ambiente de prueba a partir de dicho respaldo.
RF6	El sistema es operable por el equipo de soporte sin requerir conocimiento	Media	Se ejecuta el pipeline completo mostrando en los logs cada etapa y su resultado, sin necesidad de pasos

	profundo del código fuente, mediante parámetros claros.		manuales intermedios fuera del flujo definido.
RF7	El sistema permite agregar nuevos módulos, entornos o ajustes en los pasos del pipeline con cambios acotados y sin reescribir completamente la solución.	Media	Para una ejecución de prueba se dispone de logs estructurados donde se identifica por cliente/entorno el estado de cada paso y el tiempo consumido.
RF8	El sistema se integra con la infraestructura y herramientas existentes (SVN, Tomcat, Oracle, infraestructura cloud y herramientas de gestión del proyecto).	Alta	Al terminar una ejecución masiva se genera un resumen visible con la lista de entornos exitosos y fallidos y el paso donde ocurrió el error.
RF9	El sistema sigue buenas prácticas de DevOps y CICD (automatización, integración continua, entregas frecuentes, feedback temprano) adaptadas al contexto de SVE en la nube.	Media	En pruebas, cuando falta una propiedad obligatoria por entorno, el pipeline se detiene con mensaje de error específico; cuando están completas, la ejecución avanza sin interrupciones.

7 Análisis comparativo de sistemas y herramientas

Se preseleccionaron tres herramientas clave para el desarrollo del sistema de actualización masiva en la Suite Visión Empresarial (SVE), considerando especialmente su capacidad de integración con Subversion, que es el sistema de control de versiones utilizado actualmente en la empresa. La integración fluida con Subversion asegura la gestión centralizada del código fuente y la automatización de los procesos relacionados con la integración continua y despliegue.

7.1 Descripción de herramientas

7.1.1 Ansible Automation Platform

Es una plataforma madura y de código abierto para automatización de configuraciones y despliegues. Funciona sin agentes, es compatible con entornos Linux y nube, y permite actualizaciones seguras y escalables mediante playbooks. Tiene amplio soporte comercial y comunidad activa.

Está enfocada en la automatización integral y escalable de tareas y procesos en entornos heterogéneos. Utiliza playbooks legibles en YAML para configurar, desplegar y gestionar sistemas sin necesidad de agentes instalados. Facilita flujos de trabajo complejos, garantiza seguridad mediante control de acceso y auditorías, y se integra con plataformas cloud y de terceros para automatización completa de infraestructuras TI.

7.1.2 Jenkins

La herramienta ya en uso en la empresa para orquestar pipelines CI/CD. Permite integrarse con Subversion para la gestión centralizada de código y automatiza los procesos de construcción, prueba y despliegue. Es ampliamente usada, con alta madurez tecnológica y soporte robusto.

Su enfoque es la orquestación de pipelines CI/CD para integración y entrega continua de software. Permite automatizar la construcción, pruebas y despliegues basados en cambios en repositorios (como Subversion). Es altamente extensible con plugins, soporta múltiples lenguajes y sistemas, y centraliza el ciclo de vida del desarrollo con monitoreo y control continuo.

7.1.3 NinjaOne

Solución comercial para gestión y despliegue automatizado de software en endpoints Windows, macOS y Linux. Ofrece monitoreo en tiempo real, administración remota y alto nivel de automatización, ideal para entornos heterogéneos con soporte dedicado.

Se centra en la gestión y despliegue automatizado de software en endpoints de múltiples sistemas operativos (Windows, macOS, Linux). Ofrece administración remota, monitoreo en tiempo real y alta automatización para entornos distribuidos. Su enfoque es facilitar operaciones de TI seguras y consistentes, con soporte comercial dedicado para garantizar estabilidad y evolución adaptada a necesidades empresariales.

7.2 Criterios y Objetivos para Evaluar Herramientas

Tabla 3

Criterios y Objetivos para Evaluar Herramientas

Criterio	Indicador	Peso (1-5)	Descripción de Calificación
Compatibilidad infraestructura	Nivel de integración con infra actual (ALTO, MEDIO, BAJO)	5	5=Integración total, 3=parcial, 1=ninguna
Escalabilidad	Capacidad usuarios y procesos simultáneos	4	5=soporta crecimiento masivo, 3=limitada, 1=no escalable
Seguridad	Soporte para autenticación, control acceso, encriptación	4	5=robusto, 3=aceptable, 1=deficiente
Requisitos técnicos	Compatibilidad sistemas, recursos hardware/software	3	5=bajo requerimiento, 3=moderado, 1=alto
Mantenibilidad	Facilidad actualización y soporte	3	5=excelente documentación y

			comunidad, 1=muy limitada
Costo estimado	Costo total mensual o anual en COP/USD	4	5=gratis/open source, 3=costo moderado, 1=costo alto
Nivel de automatización	Grado de automatización en CI/CD	5	5=automatización completa, 3=parcial, 1=nula
Soporte	Disponibilidad y calidad de soporte técnico y comunidad	4	5=soporte 24/7 y comunidad activa, 1=soporte nulo o pobre

7.3 Matriz Comparativa de Herramientas

Tabla 4

Matriz Comparativa de Herramientas

Criterio	Peso	Ansible Automation Platform	Jenkins
Compatibilidad infraestructura	5	5 (Compatible con Linux, cloud, integrable)	5 (Ya integrado en infraestructura existente)
Escalabilidad	4	5 (Altamente escalable, diseñada para cloud)	4 (Escalable con plugins, depende configuración)
Seguridad	4	5 (Control acceso, auditorías integradas)	4 (Seguridad según configuración y plugins)
Requisitos técnicos	3	4 (Requiere entorno Linux, manejo YAML)	4 (Requiere configuración Java/WAR)
Mantenibilidad	3	5 (Open source, amplia documentación)	4 (Soporte comunitario y plugins)
Costo estimado	4	5 (Open source, costos infraestructura)	5 (Open source, sin licencia)
Nivel de automatización	5	5 (Automatización avanzada con playbooks)	5 (Automatización completa de CI/CD workflows)

Soporte	4	4 (Soporte Red Hat y comunidad)	4 (Comunidad amplia y soporte varias empresas)
Tiempo de aprendizaje	3	3 (Requiere aprendizaje de YAML y playbooks)	5 (Equipo con experiencia previa)
Costo aprendizaje	3	3 (Capacitación necesaria)	5 (Sin capacitación adicional importante)

7.3.1 Puntajes Totales (Peso x Calificación)

Tabla 5

Puntajes Totales (Peso x Calificación)

Tecnología	Puntajes totales
Ansible Automation Platform	177
Jenkins	187
NinjaOne	153

7.3.2 Análisis resultados matriz comparativa

Jenkins presenta la mayor ventaja para el proyecto gracias a la experiencia previa del equipo y un buen equilibrio entre todos los criterios evaluados, especialmente en compatibilidad, aprendizaje y costos adicionales. Esto facilita su rápida adopción y reduce riesgos.

Ansible Automation Platform destaca por su automatización avanzada, escalabilidad y costo accesible, aunque requiere capacitación para el equipo debido a la curva de aprendizaje del lenguaje YAML y el paradigma de playbooks.

NinjaOne ofrece una solución con fuerte soporte comercial y automatización especializada para endpoints, pero su costo de licenciamiento es elevado y no es open source, lo que puede limitar la flexibilidad y aumentar costos a largo plazo.

7.4 Evaluación de Riesgos

Para el proyecto de implementación del sistema de actualización masiva en la Suite Visión Empresarial (SVE), se realiza una evaluación cualitativa de riesgos para determinar cuál de las tecnologías seleccionadas —Ansible Automation Platform, Jenkins o NinjaOne— presenta la menor exposición a riesgos relevantes. Este análisis permite identificar, clasificar y priorizar riesgos según su probabilidad e impacto potencial, facilitando una comparación clara para tomar la mejor decisión tecnológica sin abordar aún acciones de mitigación.

Tabla 6

Evaluación de Riesgos

Riesgo/Tema	Descripción	Herramienta	Probabilidad (1-5)	Impacto (1-5)	(P x I)	Comentarios
Incompatibilidad con infra	Dificultad integración con infra	Ansible Automation	3	4	12	Usa playbooks, requiere Linux, integración cloud.
		Jenkins	1	3	3	Integrado en infra existente, familiaridad alta.
		NinjaOne	4	4	16	Depende de agentes, multiuso, más complejo.
Curva de aprendizaje alta	Dificultad para dominar la herramienta	Ansible Automation	4	3	12	YAML y playbooks pueden ser complejos para el equipo.
		Jenkins	2	2	4	El equipo ya tiene experiencia usando Jenkins.
		NinjaOne	3	3	9	Plataforma nueva para

						equipo, interfaz nueva.
Errores en automatización	Fallas en scripts o pipelines	Ansible Automation	2	5	10	Configuración compleja, riesgo de fallos críticos.
		Jenkins	1	5	5	Amplia experiencia y validación previa.
		NinjaOne	3	5	15	La gestión distribuida puede aumentar la posibilidad de fallos.
Vulnerabilidades seguridad	Exposición a accesos no autorizados	Ansible Automation	2	5	10	Necesita controles y auditorías estrictas.
		Jenkins	2	5	10	Vulnerabilidades conocidas, controles establecidos.
		NinjaOne	3	5	15	Plataforma comercial, pero depende del proveedor.
Costos imprevistos	Incremento por licenciamiento o infra	Ansible Automation	1	3	3	Open source, menor costo asociado.
		Jenkins	1	3	3	Open source, infraestructura ya establecida.
		NinjaOne	4	4	16	Licenciamiento comercial y costos asociados altos.
Dependencia soporte comercial	Riesgos por soporte limitado o discontinuo	Ansible Automation	2	3	6	Soporte comercial disponible pero no crítico.

	Jenkins	2	3	6	Comunidad activa, soporte variado.
	NinjaOne	4	3	12	Dependencia directa del proveedor de soporte.

7.4.1 Totales de riesgo (suma P x I) por herramienta

Tabla 7

Totales de riesgo (suma P x I) por herramienta

Tecnología	Total riesgos
Ansible Automation Platform	62
Jenkins	42
NinjaOne	81

7.4.2 Análisis de riesgos

Las conclusiones finales de la evaluación cualitativa de riesgos para la adopción de las tecnologías Ansible Automation Platform, Jenkins y NinjaOne son las siguientes:

1. Jenkins representa la opción con menor riesgo total cualitativo (42 puntos), lo que refleja su consolidación en la infraestructura actual, la experiencia previa del equipo y menores complejidades de integración y aprendizaje. Esto sugiere que su adopción es la opción más segura y eficiente para el proyecto en esta fase.

2. Ansible Automation Platform presenta riesgos moderados (62 puntos), principalmente asociados a la curva de aprendizaje, la integración técnica y la necesidad de dominar

configuraciones con playbooks YAML. Aunque es una plataforma madura y potente, requiere inversión en capacitación y pruebas previas para minimizar estos riesgos.

3. NinjaOne muestra el mayor nivel de riesgo cualitativo (81 puntos), atribuido a costos de licenciamiento altos, dependencia del soporte comercial, complejidad operativa por uso de agentes y mayor probabilidad de incompatibilidad con la infraestructura. Estos factores representan un desafío que debe considerarse con cuidado.

4. En todos los casos, los riesgos de seguridad y errores en automatización son críticos, con alto impacto, aunque con diferentes grados de probabilidad. Esto indica que, independientemente de la tecnología seleccionada, se debe planificar un análisis y control exhaustivo de seguridad y calidad una vez se avance a la fase de mitigación.

5. El análisis cualitativo permitió priorizar riesgos relevantes y compararlos de forma sencilla, facilitando la toma de decisiones sin necesidad de modelos complejos en esta etapa inicial del proyecto.

Estas conclusiones apoyan la recomendación de priorizar Jenkins para la implementación inicial, manteniendo en cuenta que Ansible Automation Platform puede ser una opción viable a mediano plazo, y que NinjaOne debe evaluarse con cautela dada su carga de riesgos.

7.5 Evaluación TRL

El nivel de madurez tecnológica, conocido como TRL (Technology Readiness Level), es una escala estandarizada que mide el grado de desarrollo o madurez de una tecnología específica, desde su concepción inicial hasta su implementación operativa en un entorno real.

Los TRL se representan típicamente en nueve niveles numerados del 1 al 9:

- TRL 1: Observación y reporte de principios básicos.

- TRL 2: Formulación del concepto tecnológico.
- TRL 3: Demostración experimental de conceptos críticos.
- TRL 4: Validación en laboratorio de componentes y subsistemas.
- TRL 5: Validación en entorno relevante (simulado o real).
- TRL 6: Demostración en entorno real de subsistemas o prototipos.
- TRL 7: Demostración del sistema completo en entorno operativo.
- TRL 8: Sistema completo y calificado a través de pruebas y demostraciones.
- TRL 9: Sistema probado y utilizado exitosamente en entorno operativo real.

Este marco fue desarrollado inicialmente por la NASA para evaluar tecnologías espaciales y actualmente se usa ampliamente en múltiples industrias y programas de investigación para estandarizar la evaluación y gestión de riesgos tecnológicos.

Este enfoque facilita identificar qué tan avanzadas están estas tecnologías, desde la validación preliminar de conceptos hasta su uso operativo en entornos reales, y cómo esto influye en el nivel de riesgo asociado a su implementación. Al asignar un nivel TRL específico a cada herramienta, se consigue una comprensión clara de su grado de madurez tecnológica, permitiendo tomar decisiones fundamentadas para seleccionar la opción más adecuada y segura para el proyecto.

7.5.1 Nivel TRL de cada herramienta

Tabla 8

Nivel TRL de cada herramienta

Herramienta	TRL estimado	Justificación
Ansible Automation Platform	TRL 8	Tecnología ampliamente probada y utilizada en múltiples industrias;

		plataforma consolidada y estable; equipo necesita capacitación pero madurez alta.
Jenkins	TRL 9	Plataforma establecida, con larga trayectoria; ya usada en la infraestructura actual; experiencia comprobada del equipo.
NinjaOne	TRL 6	Tecnología más reciente con buena aceptación comercial, pero con dependencia de soporte y agente; aún en fase de madurez operativa en entornos complejos.

7.5.2 Jenkins – TRL 9

Jenkins recibe TRL 9 porque es una plataforma de automatización CI/CD ampliamente consolidada y probada en producción en grandes empresas tecnológicas de clase mundial, como Amazon, Google, Facebook, Netflix y LinkedIn. Estas compañías utilizan Jenkins como componente central para orquestar pipelines complejos, automatizar despliegues y mantener alta confiabilidad operativa (TheirStack, s. f.).. Este nivel indica que Jenkins ha completado todas las fases desde la validación en laboratorio hasta el despliegue estable en entornos reales, ofreciendo estabilidad, escalabilidad y soporte constantes. Además, el equipo de PENSEMOS S.A. ya tiene experiencia previa con Jenkins, reduciendo riesgos de adopción.

7.5.3 Ansible Automation Platform – TRL 8

Ansible Automation Platform está en TRL 8 debido a su uso extendido en sectores críticos como banca y telecomunicaciones, con casos de éxito en entidades como BBVA y Telefónica. Estas organizaciones han implementado Ansible para automatizar configuraciones, despliegues y mejorar la eficiencia operativa (IT User, 2020). Aunque madura, Ansible requiere aún pruebas y

ajustes para adaptarse a la infraestructura específica de PENSEMOS S.A., además de capacitación para el equipo, lo cual refleja que, aunque está "lista para producción", necesita validación específica en el contexto del proyecto para alcanzar TRL 9.

7.5.4 NinjaOne – TRL 6

NinjaOne se ubica en TRL 6 porque, aunque tiene buena aceptación comercial y ha sido validada en entornos empresariales para gestión de endpoints y soporte remoto, su implementación en entornos complejos suele requerir pruebas piloto y soporte dedicado para asegurar integración, ajuste y escalabilidad. Esto significa que NinjaOne está en una fase en la que los componentes y subsistemas han sido validados, pero aún no se ha realizado la demostración completa del sistema en un entorno operativo equivalente al del proyecto, por lo que su madurez técnica y operativa no es total.

7.6 Selección de herramienta orquestación

Jenkins se presenta como la opción más sólida y confiable para la implementación inicial en el proyecto SVE. Su nivel de madurez tecnológica TRL 9 refleja su amplio y probado uso en grandes empresas tecnológicas como Amazon, Google y Netflix, garantizando estabilidad, escalabilidad y soporte en entornos reales y complejos. Además, la experiencia previa del equipo con Jenkins reduce riesgos asociados a la curva de aprendizaje y facilita una integración eficiente.

Ansible Automation Platform, con un TRL 8, es una alternativa madura y robusta, especialmente valorada en sectores críticos como banca y telecomunicaciones. Sin embargo, requiere etapas previas de capacitación y pruebas específicas para el contexto del proyecto, lo que implica ciertos riesgos operativos que deben ser mitigados con un plan de implementación gradual.

NinjaOne, aunque tecnológicamente viable, tiene un nivel TRL 6, indicando que aún está en proceso de validación dentro de entornos empresariales complejos como SVE. Su dependencia del soporte comercial y necesidad de pruebas piloto para garantizar su operatividad en la infraestructura objetivo representan riesgos adicionales que hacen menos recomendable su adopción inmediata.

En resumen, la elección de Jenkins maximiza la probabilidad de éxito al combinar alta madurez tecnológica, menor riesgo operativo y experiencia comprobada, asegurando así un despliegue efectivo y confiable para el sistema de actualización masiva. La adopción de Ansible puede considerarse en fases futuras una vez confirmadas las capacidades del equipo y superados los procesos de validación. NinjaOne deberá seguir evaluándose con pruebas piloto antes de cualquier implementación formal.

8 Revisión de arquitectura y herramientas CI/CD para la solución

La selección priorizó tecnologías existentes en Pensemos S.A. para minimizar riesgos de adopción, garantizar compatibilidad con SVE y cumplir requisitos funcionales/no funcionales criterios clave:

Compatibilidad: Integración nativa con SVN, Tomcat y Oracle de SVE.

Escalabilidad y rendimiento: Soporte para pipelines multi-entorno (RNF2).

Mantenibilidad: Uso de herramientas maduras con soporte interno (RNF8).

Costo: Explotación de licencias existentes (OCI, Jenkins open source).

Facilidad de uso: Curva de aprendizaje baja para equipo de soporte.

Tabla 9

Revisión de arquitectura y herramientas CI/CD para la solución

Tecnología	Versión/Especificación	Rol en el sistema	Justificación
Java	JDK 1.8 (con opciones JVM como -Xmx4096m)	Entorno de ejecución para Ant y tasks custom (VEAntTask).	Base de SVE y herramientas internas; compatible con ojdbc6/8 para Oracle.
Subversion (SVN)	SVN 1.8+ con tasks SveSvnAnt	Checkout y control de versiones de empaquetado/módulos.	Repositorio oficial de PENSEMOS S.A.; tasks custom para info de ramas
Apache Ant	Con VEAntTask y contribuciones	Compilación, empaquetado WAR y ejecución de scripts de despliegue.	Estándar en procesos SVE; extensible con tasks propietarias de PENSEMOS S.A.
Apache Tomcat	Versiones soportadas (6+), con directorios lib	Servidor de aplicación para WARs de SVE.	Plataforma de despliegue nativa de SVE; stop/start vía SSH.
Oracle DB	Con ojdbc6.jar y ojdbc8.jar	Actualizaciones de esquemas con scripts y VeUpdateSuite.	BD oficial de SVE; conectores JDBC compatibles.

8.1 Herramientas y métodos propietarios de Pensemos S.A.

El sistema aprovechara métodos existentes en el ecosistema SVE para control de versiones y empaquetado:

VEAntTask: Framework custom de Ant para compilación modular de SVE (genera WARs según módulos listados en propiedades).

SveSvnAnt (ve.task.SveSvnAnt): Task Def custom que ejecuta comandos SVN (info) para obtener versiones de proyectos como Empaquetado, Scripts, AgentApi, VeUpdateSuite, SveInstaller, WebSite, VEFile, Templates, Revo Pocket, Utils. Ejemplo de targets:

<target name="info">

```
    <taskdef name="svesvn" classname="ve.task.SveSvnAnt" classpathref="vetask" />
    <svesvn command="info" dir="${basedir}" prefix="${svn.prefix}" />
</target>
```

Usado en targets como infoPacking, infoScripts, infoVeUpdateSuite para validar ramas SVN antes de checkout (RNF6).

VeUpdateSuite: Herramienta propietaria para actualización de BD.

VEInfo/VETask: Métodos para introspección de versiones SVN en build process.

8.2 Librerías del empaquetado

El classpath de VEAntTask incluye librerías críticas para SVN, Tomcat, Oracle y análisis estático, ubicadas en \Empaquetado\lib:

Tabla 10

*Librerías del empaquetado***.**

Librería principal	Rol	Directorio asociado
ojdbc6.jar, ojdbc8.jar	Conectores JDBC para Oracle BD (actualizaciones y backups).	Oracle
jsch-0.1.55.jar	SSH para despliegue remoto en Tomcat/servidores.	SSH
mysql-connector-java-5.0.8-bin.jar, sqljdbc4.jar	Soporte BD alternativos (MySQL/SQL Server).	BD
ant-contrib-0.6.jar	Extensiones lógicas para Ant (loops, condicionales en VEAntTask).	Ant
commons-net-3.3.jar	Protocolos de red (FTP/SSH para distribución).	Networking
checkstyle-all-5.0.jar, pmd-4.2.5.jar	Análisis estático de código en compilación.	Calidad
lombok.jar	Generación automática de boilerplate en Java.	Java

El uso de estas herramientas permite: Integración nativa: Aprovecha el stack existente (SVN, Tomcat, Oracle) sin migraciones disruptivas. Tasks propietarias (SveSvnAnt, VEAntTask) ya integradas en flujos SVE.

Control granular: SveSvnAnt ejecuta svn info en proyectos específicos (Empaquetado, VeUpdateSuite, WebSite, etc.) obteniendo versiones exactas por prefix (RNF6).

Costo cero: VEAntTask genera WARs dinámicos según los modulos. (hasta 1.5GB), sin reempaquetado completo (RF10). Costo total de propiedad: Cero licencias. Jenkins open source + herramientas internas ya licenciadas.

9 Desarrollo iterativo

El desarrollo de la solución se realizó con una arquitectura de automatización basada en build_deploy.xml, implementando un pipeline build + deploy para la Suite Visión Empresarial en modo standalone. Este pipeline fue diseñado para resolver el contexto de versión desde SVN, preparar las herramientas de build, construir los artefactos requeridos y ejecutar la actualización de uno o varios ambientes destino con trazabilidad completa de tiempos y fases.

La implementación siguió un enfoque incremental, donde cada bloque del flujo se encapsuló en targets pequeños y reutilizables, permitiendo validar de manera independiente la obtención de versiones, el checkout de herramientas, la compilación de componentes, la generación del reléase y la actualización por entorno. De esta forma, el sistema no se construyó como un bloque monolítico, sino como una secuencia de componentes orquestados y ajustados progresivamente según las necesidades reales de operación. Posteriormente, en la fase de documentación, se presentará el funcionamiento de cada paso del sistema con mayor nivel de

detalle, explicando la responsabilidad técnica de cada target y su interacción dentro del flujo completo.

9.1 Desarrollo y codificación por incrementos funcionales

El desarrollo de la solución se realizó con una arquitectura de automatización basada en build_deploy.xml, implementando un pipeline de build + deploy para la Suite Visión Empresarial en modo standalone. Como parte inicial de esta fase se realizó el montaje del entorno de integración continua en Jenkins, incluyendo la configuración del Job principal, la definición de variables y propiedades de ejecución, la vinculación con el repositorio SVN y la habilitación de los plugins requeridos para la orquestación del pipeline. En particular, se configuraron componentes como el plugin de Ant, el plugin de Subversion y el entorno JDK necesario para ejecutar el proceso de construcción y despliegue.

El desarrollo se estructuró en incrementos funcionales que corresponden a las fases principales del pipeline. En la primera etapa se implementó el Bootstrap del proceso, con la inicialización de tiempos mediante init-timing, la impresión del entorno de ejecución con print-versions y la resolución del contexto de marca SVN mediante infoPacking-svn-standalone y get-mark-version. Esta fase permitió establecer una referencia consistente de versión, rama y contexto antes de ejecutar cualquier operación de construcción o despliegue.

En la segunda etapa se desarrolló la capa de tooling, que incluye el checkout de VEAntTask, su compilación y preparación como JAR local, y el registro de tareas personalizadas de Ant cómo BuildAllModules, SveSvnAnt y LastFinalVersion. Este bloque fue necesario para extender Apache Ant con lógica específica del ecosistema SVE, incluyendo consultas SVN especializadas, orquestación por módulos y resolución de la última versión liberada.

En la tercera etapa se incorporó la lógica de checkout masivo y empaquetado, donde el pipeline descarga todos los módulos requeridos del release, obtiene metadatos de SVN por módulo y construye el paquete fuente de distribución. Finalmente, se añadió el deploy por ambiente, que ejecuta un loop controlado sobre updated.environments.amount, normaliza las propiedades de cada entorno y aplica el ciclo completo de actualización: undeploy del sitio, stop de Tomcat, backup, actualización de base de datos, start de Tomcat y deploy del sitio.

Este diseño incremental permitió separar responsabilidades entre targets, reducir acoplamiento y facilitar la evolución del flujo de despliegue sin afectar el comportamiento global del pipeline.

9.2 Gestión y seguimiento por incrementos

Cada incremento fue gestionado como una unidad funcional verificable, con su propio punto de entrada, validaciones internas y salida trazable en consola. Para controlar el avance, se introdujeron marcas de tiempo por fase, validaciones de disponibilidad de herramientas, verificación de archivos requeridos y mensajes contextualizados que facilitan el diagnóstico cuando un bloque falla. Esto permitió que el desarrollo pudiera avanzar de manera controlada incluso cuando se incorporan componentes con dependencia directa entre sí, como el JAR de VEAntTask, los archivos de repositorios o las propiedades por ambiente.

El seguimiento también se apoyó en el uso de trycatch en bloques críticos del despliegue por entorno, así como en validaciones tempranas con fail unless para asegurar la presencia de propiedades obligatorias como URLs de Tomcat, credenciales JDBC, prefijos y datos de conexión. Adicionalmente, el pipeline registra información como paso1.start.time, paso4.start.time,

paso7.start.time y sus respectivos cierres, lo que permite reconstruir la ejecución por bloques y evaluar el costo temporal de cada parte del proceso.

Gracias a este mecanismo, el sistema fue evolucionando sin perder trazabilidad, y cada avance quedó vinculado a una fase concreta del pipeline, lo que facilitó tanto la depuración como la validación técnica de los resultados.

9.3 Revisión y ajuste periódico del flujo de trabajo para optimizar eficiencia

Durante la construcción del pipeline se realizaron ajustes periódicos orientados a mejorar la eficiencia operativa y la robustez del flujo. Uno de los cambios más relevantes fue la incorporación de una lógica que evita reconstrucciones innecesarias, verificando primero si el release ZIP ya existe antes de ejecutar de nuevo el build pesado. Esta optimización redujo tiempo de ejecución y permitió reutilizar artefactos ya generados cuando el contexto de versión no había cambiado.

Otro ajuste importante fue la consolidación de un esquema de actualización por ambiente con normalización de propiedades, lo que permitió reutilizar la misma lógica interna para diferentes entornos sin duplicar código. El mapeo de propiedades upd.environmentN.* a variables generales de ejecución hizo posible que los subtargets operaran de forma homogénea, independientemente del número de ambientes configurados. Además, se incorporaron mecanismos de recuperación y diagnóstico, como la limpieza de working copies SVN, el respaldo previo de bases de datos y la impresión de la fase activa cuando ocurre un error.

Estos ajustes fortalecieron la estabilidad del flujo, redujeron puntos de fallo y mejoraron la observabilidad del proceso completo. Como resultado, el pipeline quedó estructurado como un

orquestador modular, resiliente y trazable, capaz de ejecutar build y deploy de manera repetible sobre múltiples entornos.

10 Retroalimentación y ajuste

La fase de retroalimentación y ajuste se desarrolló como un mecanismo de revisión posterior a cada incremento funcional, con el objetivo de validar el comportamiento del sistema, identificar oportunidades de mejora y redefinir prioridades para los siguientes pasos. Esta etapa permitió evaluar no solo el funcionamiento técnico del pipeline, sino también su claridad operativa, la consistencia de los mensajes de consola, la trazabilidad de las ejecuciones y la facilidad de uso para el equipo encargado de la operación.

A partir de esta revisión se consolidaron ajustes sobre la estructura del flujo, la parametrización del proceso y la forma en que se documentan y presentan las etapas del sistema. De esta manera, la retroalimentación no se limitó a corregir errores, sino que también sirvió para refinar la solución y orientar la evolución del desarrollo hacia una versión más estable, comprensible y mantenible.

10.1 Revisión y evaluación formal de cada incremento

Cada incremento implementado fue sometido a una evaluación formal antes de considerarse estable e integrado al flujo general del sistema. Esta revisión consistió en verificar el comportamiento del target o bloque correspondiente, observar su ejecución en consola, validar el cumplimiento de los prerrequisitos definidos y confirmar que el resultado obtenido fuera coherente con el propósito funcional esperado.

En esta etapa también se revisaron aspectos como el tiempo de ejecución de cada fase, la correcta propagación de propiedades, la disponibilidad de artefactos intermedios y la respuesta del pipeline frente a errores o validaciones fallidas. Este control permitió identificar qué partes del proceso estaban suficientemente maduras y cuáles requerían ajustes adicionales antes de continuar con la siguiente iteración.

10.2 Recopilación de retroalimentación del equipo

La retroalimentación del equipo se obtuvo a partir de la revisión conjunta de las ejecuciones, la observación de los logs y el análisis de los mensajes generados por Jenkins y por los targets del pipeline. Esta información permitió detectar si los pasos del sistema resultaban suficientemente claros para operación, si los errores quedaban bien contextualizados y si la secuencia de despliegue facilitaba el diagnóstico de fallas.

Además, la revisión con el equipo ayudó a identificar necesidades de ajuste en la documentación interna, en la parametrización por entorno y en la estructura general del flujo, especialmente en los puntos donde se requería mayor claridad operativa. Gracias a esa retroalimentación, el desarrollo se mantuvo alineado con las condiciones reales de uso y con las expectativas del equipo responsable de ejecutar y mantener la solución.

11 Documentación técnica

El sistema de actualización masiva para la Suite Visión Empresarial (SVE) adopta una arquitectura de orquestación centralizada basada en pipelines CICD, donde Jenkins actúa como el núcleo coordinador de todas las fases del proceso, integrando herramientas establecidas en el ecosistema de Pensemos S.A. para garantizar compatibilidad y mínima disrupción operativa.

Esta arquitectura se organiza en capas funcionales: la capa de repositorios (SVN) proporciona los artefactos fuente; la capa de construcción (VEAntTask y Apache Ant) genera los paquetes WAR parametrizables; la capa de orquestación (Jenkins) válida prerrequisitos, ejecuta backups y coordina despliegues multi-entorno; y la capa de ejecución (servidores Tomcat y bases de datos Oracle en la nube) recibe las actualizaciones de forma controlada.

El diseño prioriza la automatización de procesos repetitivos, la parametrización para flexibilidad multi-cliente y mecanismos de validación temprana para cumplir con los requisitos funcionales (RF1–RF10) y no funcionales (RNF1–RNF9) definidos previamente, alineándose con los principios DevOps de entrega continua y colaboración entre desarrollo y soporte.

Componentes principales

El sistema integra los siguientes componentes clave, diseñados para soportar actualizaciones masivas de bases de datos y aplicaciones WAR en entornos cloud distribuidos:

Jenkins Pipeline como orquestador central: Gestiona la ejecución secuencial y paralela de las fases del proceso, desde la validación de propiedades hasta la notificación final, permitiendo reintentos y monitoreo en tiempo real (RF7, RNF9).

Repositorios SVN: Fuente única de empaquetado consolidado por el equipo de desarrollo, con checkout controlado para garantizar la versión correcta de módulos (RF7, RNF6).

VEAntTask y Apache Ant: Motor de compilación y empaquetado que genera WARs modulares según propiedades de configuración (RF9, RNF8).

Sistema de propiedades parametrizadas: Archivo de configuración que define módulos, credenciales, rutas de despliegue, cantidad de entornos y prefijos, con validación automática de campos obligatorios (RF6, RF9).

Mecanismos de despliegue SSH y Tomcat: Scripts que detienen Tomcat, despliegan WAR y reinician el servicio por entorno, integrados al pipeline (RF4).

Motor de actualización de base de datos: Ejecución remota de scripts de versión y VeUpdateSuite, precedida de backup (RF3, RF5).

11.1 Modelo de configuración

La configuración se basa en un archivo de propiedades de ant, único que parametriza el pipeline entero:

Propiedades:

- modules: Lista de módulos SVE (ej: base,ind,bsc,...).
- updated.environments.amount: Número de entornos a actualizar (ej: 2).
- environmentN.tomcat.host/user/pwd: Credenciales SSH Tomcat.
- environmentN.db.host/user/pwd: Credenciales Oracle BD.

Este diseño asegura flexibilidad (RNF8) y validación temprana (RF6), con mensajes de error claros si faltan campos críticos (esto es solo un ejemplo, más adelante se mostraran a detalle cada propiedad).

11.2 Mecanismos de control de errores y recuperación

El diseño incorpora recuperación ante fallos en múltiples niveles:

- Validación de prerrequisitos: Detiene si no hay acceso o propiedades (RF6).
- Backup previo a BD: Permite rollback de base de datos si falla la actualización a la base de datos (RF5, RNF3).
- Reintentos: Pipeline soporta ejecución selectiva por entorno fallido.
- Notificaciones: Resumen final en consola de Jenkins con estados y errores (RF9).

11.3 Flujo del pipe

La Suite Visión Empresarial ahora cuenta con un proceso de empaquetado y despliegue estructurado que va desde la consolidación del código en SVN hasta la actualización automática de uno o varios entornos de aplicación. A continuación se presenta una versión del flujo, pensada como documentación interna detallada para desarrollo, QA, soporte e infraestructura.

11.3.1 Visión general del pipeline

El objetivo del pipeline es tomar una rama estable de empaquetado de la Suite Visión Empresarial, construir el paquete completo (código, fuentes y artefactos de actualización) y desplegarlo secuencialmente en los entornos configurados.

El proceso está dividido en pasos numerados (PASO 0 al PASO 7), cada uno con tiempos de inicio/fin, mensajes claros en consola y validaciones específicas, lo que facilita el diagnóstico y la auditoría del ciclo de despliegue.

11.3.2 PASO 0 - Checkout inicial del empaquetado en Jenkins

El job de Jenkins está configurado para usar Subversion como origen de código, apuntando a la URL del repositorio de empaquetado (por ejemplo, la rama branches / SVE_5_7_1_mhcp / Empaquetado), autenticado con las credenciales definidas para el usuario de integración (por ejemplo, yibarra).

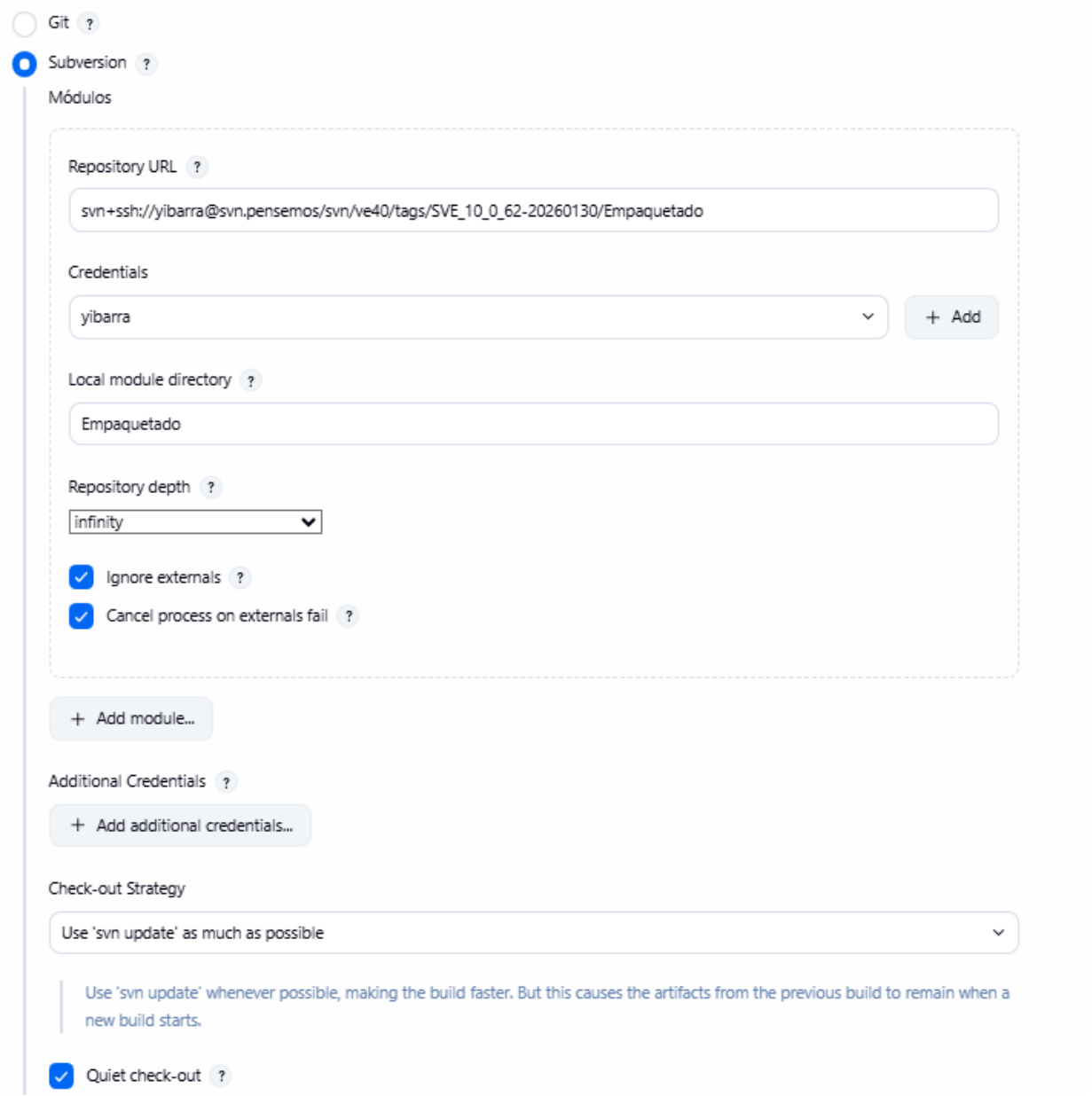
Antes de invocar Ant, Jenkins ejecuta la estrategia de checkout configurada (por ejemplo, Use 'svn update' whenever possible), lo que permite reutilizar la copia de trabajo cuando no hay cambios, acelerando el proceso, o traer una copia limpia cuando es necesario.

El resultado de este paso es que el workspace del job contiene el directorio Empaquetado actualizado a la revisión más reciente, sobre el cual luego se ejecuta el comando Ant build-and-deploy-standalone con todos los parámetros de entornos, marcación y credenciales.

Si el checkout falla (por errores de red, credenciales o configuración de repositorio), el job se detiene antes de llegar al PASO 1 del flujo Ant, ya que no existiría una copia válida del empaquetado desde la cual continuar.

Figura 2

Paso 0 Checkout inicial



11.3.3 PASO 1 – Información SVN del empaquetado

En este primer paso, el proceso toma como entrada el contexto de ejecución suministrado por la herramienta de integración continua, incluyendo la URL del repositorio, la rama activa y la versión de la Suite que se está empaquetando. A partir de esa información se construye el contexto base del empaquetado, que será utilizado por el resto del flujo para asegurar coherencia entre la rama de trabajo, la versión activa y los componentes auxiliares involucrados en la ejecución.

Durante esta etapa se definen propiedades como packing.svn.version, packing.svn.basedir y packing.svn.branch, las cuales permiten identificar de forma explícita la versión que se va a procesar, el módulo base al que pertenece el flujo y el tipo de rama sobre el que se está trabajando. Con estas variables se genera además una traza consolidada del estado del empaquetado, por ejemplo: Base Directory: Empaquetado, Versión: SVE_5_7_1_mhcp, Branch Type: branches, Full Info: Empaquetado-SVE_5_7_1_mhcp-branches.

Este paso garantiza que tanto el empaquetado principal como las herramientas auxiliares, como VEAntTask, operen sobre la misma referencia de código, evitando inconsistencias entre la versión declarada y la realmente utilizada en el checkout o en la construcción de artefactos.

Registro del inicio del paso (paso1.start.time).

El target ejecuta un bloque tstamp para registrar la hora exacta de inicio del PASO 1. Este dato no cumple solo una función informativa, sino que permite medir la duración real del paso y facilitar la auditoría de una ejecución en caso de análisis de fallos.

Evaluación de propiedades de entrada de marcación.

El target verifica si existen propiedades de marcación suministradas por Jenkins o por archivos de configuración, en especial mark.version.from y mark.from. Esta validación determina la fuente desde la cual se tomará la referencia de versión que guiará el flujo posterior.

Resolución de current.branch.name.

Si mark.version.from está definido, se utiliza ese valor como nombre de versión activa; en caso contrario, se asigna un valor por defecto. Este mecanismo evita que el proceso quede sin una versión válida cuando el job se ejecuta sin todos los parámetros esperados.

Asignación de packing.svn.version.

Una vez resuelto current.branch.name, el valor se copia a packing.svn.version. Desde este momento, esta propiedad se convierte en la variable canónica de versión para el resto del pipeline.

Fijación del directorio base lógico (packing.svn.basedir=Empaquetado).

El target establece de forma explícita que el módulo base del flujo es Empaquetado. Esta propiedad se utiliza para construir mensajes de trazabilidad y para mantener coherencia entre logs, artefactos y componentes descargados.

Resolución del tipo de rama (packing.svn.branch).

Se define inicialmente un valor por defecto, por ejemplo branches, y luego se evalúa mark.from para sobrescribirlo si existe. De este modo, Jenkins puede forzar una estrategia de ejecución sobre trunk, branches o tags, según el tipo de release requerido.

Consolidación y publicación de la información de contexto en el log.

Con packing.svn.basedir, packing.svn.version y packing.svn.branch ya definidos, el target imprime un resumen consolidado del estado SVN del empaquetado. Este bloque constituye la evidencia principal para verificar que el pipeline trabajará sobre la referencia correcta.

Registro de cierre del paso (paso1.end.time).

Finalmente, se registra la hora de fin y se imprime el rango completo de ejecución del paso. Esto deja trazabilidad suficiente para calcular tiempos y para integrar esta etapa dentro del resumen global del pipeline.

Al finalizar este target deben quedar definidas y consistentes, como mínimo, las siguientes variables:

current.branch.name.

packing.svn.version.

packing.svn.basedir.

packing.svn.branch.

paso1.start.time.

paso1.end.time.

Propiedades relevantes

mark.version.from.

mark.from.

packing.svn.version.

packing.svn.branch.

packing.svn.basedir.

Salida funcional

El resultado funcional de este paso es la consolidación de un contexto de versión estable para que el pipeline realice el checkout y la ejecución de VEAntTask sobre la referencia correcta del código fuente.

Evidencias de log esperadas

Inicio del PASO 1 con timestamp.

Resumen de Base Directory, Version y Branch Type.

Cierre del PASO 1 con rango horario de ejecución.

Criterio de aceptación

El paso se considera correcto cuando en el log se observa la línea de información completa construida a partir de Empaquetado, la versión resuelta y el tipo de rama correspondiente.

Si quieres, en el siguiente mensaje te lo puedo convertir a un estilo todavía más “de libro” y menos técnico-operativo, o dejarlo con formato de subsección numerada para que encaje directo en tu capítulo.

11.3.4 PASO 2 – Checkout de herramientas VEAntTask

Este paso tiene como finalidad descargar el componente VEAntTask para habilitar las tareas personalizadas de Ant utilizadas en la construcción, el checkout masivo y los ciclos iterativos del pipeline.

Descripción de VEAntTask en el flujo

Apache Ant permite ejecutar tareas como unidades funcionales dentro de un proceso de construcción. Estas tareas representan acciones concretas que el motor de build puede invocar, como copiar archivos, ejecutar comandos, compilar código, consultar repositorios o iterar sobre conjuntos de elementos. Además de sus tareas nativas, Ant admite la incorporación de tareas personalizadas mediante mecanismos de registro como taskdef, lo que amplía sus capacidades para adaptarse a necesidades específicas de un proyecto.

En este contexto, VEAntTask corresponde al conjunto de extensiones personalizadas desarrolladas para la suite SVE. Se trata de una librería de clases Java que incorpora funcionalidades que no están disponibles de forma estándar en Ant y que resultan necesarias para

ejecutar operaciones especializadas dentro del pipeline. Gracias a este componente, el flujo puede realizar acciones como consultas y checkout SVN especializados, construcción masiva de módulos, iteración controlada de ambientes y propiedades, y resolución de versiones liberadas a partir de archivos de control.

Por esta razón, el PASO 2 no constituye un checkout accesorio, sino la descarga del componente que habilita la lógica operativa del pipeline. En otras palabras, el proceso no utiliza únicamente Ant como motor de automatización, sino una versión extendida de Ant soportada por VEAntTask para ejecutar reglas de negocio relacionadas con integración y despliegue.

Integración técnica de VEAntTask en Ant

La integración de VEAntTask dentro del proceso se realiza en una secuencia definida:

1. descarga de VEAntTask en este paso,
2. compilación y publicación del componente como un JAR local en el paso siguiente,
3. registro de las tareas mediante taskdef usando el classpath asociado a vetask,
4. invocación de dichas tareas dentro de los targets responsables del checkout, el versionado y la actualización por entorno.

Este mecanismo permite desacoplar el script principal de implementación de la lógica especializada. Así, el archivo XML de Ant se encarga de orquestar el flujo general, mientras que VEAntTask encapsula el comportamiento técnico específico requerido por la suite.

Flujo interno detallado

1. Resolución de branch.version para VEAntTask.

El target compara packing.svn.branch con packing.svn.version. Si ambos valores coinciden, interpreta que la descarga debe realizarse desde trunk; en caso contrario, construye la ruta como <branch>/<version>. Esta decisión define con precisión el árbol del repositorio desde el cual se obtendrá la herramienta.

2. Construcción de repository.URL en el repositorio common.

Una vez definido branch.version, el proceso compone la URL final de SVN para VEAntTask usando svn.repository.user y el identificador ve.ant. En esta etapa queda determinado el endpoint real desde el cual se descargará el componente.

3. Emisión de trazas de depuración previas al checkout.

Antes de ejecutar la descarga, el sistema imprime en el log los valores de packing.svn.branch, packing.svn.version, branch.version y ve.ant. Estas trazas permiten validar rápidamente que Jenkins está apuntando al árbol correcto sin necesidad de inspección manual.

4. Validación de disponibilidad del cliente SVN en el classpath.

Se verifica la presencia de la clase org.tmatesoft.svn.cli.SVN dentro de packLib. Si la clase no está disponible, el target no continúa con el checkout y detiene el proceso con un fallo controlado, evitando errores ambiguos en etapas posteriores.

5. Preparación de parámetros de autenticación y ejecución no interactiva.

Cuando el cliente SVN está disponible, el target invoca la clase correspondiente en modo fork, utilizando credenciales SSH, el comando co, y las banderas --no-auth-cache y --non-interactive. Esta configuración garantiza una ejecución reproducible en Jenkins, sin interacción manual ni almacenamiento local de credenciales.

6. Ejecución efectiva del checkout de VEAntTask.

El comando descarga VEAntTask en el workspace esperado, normalmente como un directorio hermano de Empaquetado. Si el proceso finaliza correctamente, queda evidencia del éxito en consola y se habilita el paso de compilación del JAR vetask.

7. Rama de error cuando el cliente SVN no está disponible.

Si la clase SVN no se encuentra, el target imprime el classpath efectivo para facilitar el diagnóstico y lanza un fallo explícito. Este comportamiento permite identificar de inmediato problemas de dependencias o configuración, en lugar de detectarlos más adelante por efectos secundarios.

Resultado técnico del paso

Al finalizar correctamente este target, deben cumplirse las siguientes condiciones:

1. branch.version queda resuelta según la estrategia trunk, branches o tags.
2. repository.URL queda construida y visible en el log.
3. El cliente SVN queda validado en el classpath como org.tmatesoft.svn.cli.SVN.
4. La copia de trabajo de VEAntTask queda descargada en el workspace.
5. El pipeline queda habilitado para compilar VEAntTask en el PASO 3.

Dependencias técnicas

1. Librerías SVN ubicadas en lib/svn*.jar.
2. packLib correctamente resuelto.
3. Propiedades svn.repository.user, svn.private.key.file y passphrase definidas.

Evidencias de log

1. Depuración de packing.svn.branch y packing.svn.version.
2. URL de checkout construida.

3. Mensaje de éxito del checkout de VEAntTask.

11.3.5 PASO 3 – Compilación y preparación de VEAntTask

Este paso transforma el código fuente descargado en el PASO 2 en un componente operativo para el proceso de construcción. En otras palabras, pasa de disponer únicamente de las fuentes de VEAntTask a contar con un runtime utilizable por el pipeline para ejecutar tareas custom de forma homogénea y repetible.

Sin esta compilación, el flujo no podría registrar ni ejecutar tareas personalizadas en etapas posteriores, como los checkouts masivos, la resolución de versiones, los loops por ambiente o las operaciones SVN específicas.

Descripción de VEAntTask en este paso

VEAntTask no se usa aquí como simple fuente Java, sino como la base funcional que habilita el comportamiento extendido del pipeline. Una vez compilado, este conjunto de clases se convierte en un JAR local que puede ser referenciado por Ant mediante taskdef y classpathref, permitiendo que el script principal invoque tareas especializadas sin incorporar toda la lógica directamente en el XML.

Flujo interno detallado por target

1. Target infoVeTask-standalone

Este subtarget establece la identidad de VEAntTask usando el contexto ya resuelto para Empaquetado. En la práctica:

1. toma packing.svn.version y packing.svn.branch,

2. los replica en vetask.svn.version y vetask.svn.branch,
3. define vetask.svn.basedir=VEAntTask,
4. deja trazabilidad de base, versión y rama en consola.

El resultado de este subpaso es que el pipeline sabe con precisión qué variante de VEAntTask va a compilar y preparar.

2. Target createVETask-standalone

Este target concentra la preparación del entorno de compilación y la generación del artefacto final.

Carga de configuración local

Antes de compilar, verifica si existe un archivo de propiedades locales dentro del directorio de VEAntTask, por ejemplo local.properties. Si el archivo está presente, lo carga con prefijo vetask, lo que permite aplicar ajustes específicos del entorno de construcción sin modificar el build principal de Empaquetado.

Saneamiento del directorio de distribución

Luego limpia el directorio de distribución de VEAntTask y lo recrea desde cero. Esta acción evita contaminación por JARs viejos, clases obsoletas o bibliotecas residuales de ejecuciones anteriores, asegurando que el resultado sea determinista.

Ejecución de compilación y empaquetado

Después de preparar el entorno, se ejecuta el build del proyecto VEAntTask, específicamente el target encargado de generar el artefacto final. Este proceso compila las clases

Java que contienen las tareas personalizadas y produce el JAR principal de distribución, normalmente nombrado como vetask-<versión>.jar.

Copia de bibliotecas auxiliares

Una vez generado el artefacto, se copian las dependencias auxiliares desde el directorio lib de VEAntTask hacia el directorio de distribución local. Durante esta copia se excluyen librerías estándar redundantes, como ant.jar, para evitar conflictos con el runtime ya disponible en Jenkins.

El resultado de este target es una distribución local autocontenida, lista para ser utilizada por los pasos posteriores del pipeline.

3. Target print-vetask-version

Este subtarget cierra el proceso imprimiendo en consola un resumen operativo de la versión detectada, la rama, el directorio base y la ruta esperada del JAR generado. Su función no es solo informativa: también actúa como punto de verificación para confirmar que el pipeline preparó la versión correcta de VEAntTask.

Flujo interno detallado dentro del createVETask-standalone

1. Verificación del archivo build.xml de VEAntTask.

Si no existe, el proceso falla de forma explícita con un mensaje claro, evitando que el pipeline continúe en un estado inconsistente.

2. Ejecución del target de compilación con control de propiedades.

El build se lanza con propiedades controladas para limitar el paso de información innecesaria desde el proceso padre y reducir efectos colaterales.

3. Definición de dist.dir hacia vetask.distdir.

La salida del empaquetado se dirige a la ruta esperada por el classpathref vetask, que será usada por los pasos posteriores.

4. Asignación de vetask.svn.version para trazabilidad.

El nombre del artefacto queda alineado con la versión que se está procesando, lo que facilita identificar qué distribución fue generada en cada ejecución.

5. Copia de librerías de soporte y construcción de la distribución final.

La distribución queda preparada para registrar tareas personalizadas sin depender de librerías faltantes.

Resultado técnico del paso

Al finalizar correctamente este paso, deben cumplirse las siguientes condiciones:

1. vetask.svn.version, vetask.svn.branch y vetask.svn.basedir quedan definidos.
2. El directorio vetask.distdir se recrea limpio en la corrida actual.
3. Se genera el JAR principal vetask-<versión>.jar.
4. Las librerías auxiliares se copian al directorio de distribución, excluyendo ant.jar.
5. El classpathref vetask queda listo para taskdef en los pasos posteriores.
6. El resumen de versión de VEAntTask queda visible en el log.

Artefacto clave generado

- vetask/vetask-<version>.jar

Artefactos complementarios generados

1. Bibliotecas auxiliares copiadas a vetask.distdir.
2. Metadatos de versión impresos en el log para trazabilidad.

3. Contexto de propiedades vetask.* disponible para targets dependientes.

Importancia del paso

Este paso es crítico porque sin el JAR generado no existen las tareas personalizadas que usa el pipeline, como:

1. ve.task.BuildAllModules
2. ve.task.SveSvnAnt
3. ve.task.TargetLoop
4. ve.task.LastFinalVersion

Además, si la distribución de VEAntTask no queda correctamente preparada, el pipeline puede fallar en tiempo de ejecución por errores de carga de clases o por un taskdef incompleto, incluso si el checkout anterior fue exitoso.

11.3.6 PASO 4 – Checkout de todos los módulos de la Suite

Este paso convierte una referencia de versión global en un conjunto real de working copies locales para toda la suite. En otras palabras, pasa de tener una versión objetivo a disponer físicamente de todos los módulos en disco para poder compilar, empaquetar y actualizar el sistema.

A nivel funcional, este es el punto en el que se reduce al mínimo el riesgo de deriva entre módulos, ya que todos los proyectos se alinean con la misma estrategia de rama, trunk o tags definida al inicio del pipeline. La operación de checkout se realiza de forma no interactiva y con mecanismos de limpieza previa para evitar inconsistencias en working copies ya existentes.

Interpretación práctica del paso

Este flujo puede entenderse como un proceso organizado en tres capas:

1. Capa 1: catálogo maestro.

repositories.txt define qué módulos deben procesarse.

2. Capa 2: catálogo por módulo.

3. Cada módulo apunta a un archivo <modulo>_projects.txt con los proyectos concretos que deben descargarse.

4. Capa 3: ejecución por proyecto.

Cada proyecto se descarga desde la ruta SVN correspondiente y queda listo en disco.

Este diseño evita checkouts manuales proyecto por proyecto. En su lugar, el pipeline interpreta listas de configuración y ejecuta una descarga consistente, repetible y centralizada.

Al terminar este paso, el workspace ya no contiene únicamente Empaquetado y VEAntTask, sino también todos los directorios de proyectos requeridos para construir la suite completa. Esto es fundamental porque los pasos posteriores asumen que dichos módulos existen localmente y pueden ser referenciados mediante rutas relativas.

Si uno de los proyectos falla, el estado final queda incompleto aunque la mayoría de módulos se hayan descargado correctamente. Por ello, desde el punto de vista de calidad operativa, este paso debe tratarse como una unidad crítica: la corrida solo se considera confiable si el conjunto completo de módulos queda disponible.

Qué válida buildAll en este contexto

En este paso, buildAll no realiza compilación. Su función es actuar como orquestador del checkout:

1. lee archivos de módulos,

2. itera proyectos,

3. invoca targets auxiliares,
4. delega en svesvn la ejecución concreta de SVN.

Es decir, buildAll controla el flujo y svesvn ejecuta la operación de checkout.

Flujo interno

Apertura del paso y trazabilidad temporal. Se registra paso4.start.time para medir la duración del checkout masivo y correlacionar tiempos con congestión de red, bloqueos SVN o tamaño de los módulos.

Validación del prerequisito principal: VEAntTask operativo. Se verifica que exista el JAR vetask-<version>.jar. Si no existe, el checkout masivo no inicia, porque las tareas personalizadas necesarias para buildAll y svesvn no pueden registrarse. Registro de tareas custom requeridas por el checkout masivo. Se declara taskdef para:

svesvn, encargado de operaciones SVN especializadas.

buildAll, encargado de la orquestación por archivo de módulos y proyectos.

Resolución del archivo maestro de repositorios. Se compone repositoriesFile a partir de repositories.dir y repositories.file, típicamente conf/int/repositories.txt. Este archivo define qué módulos deben procesarse y en qué orden lógico.

Ejecución de buildAll con el target repository-projects. buildAll lee el catálogo maestro de módulos y, para cada entrada, dispara el target repository-projects. Ese target delega el detalle de cada proyecto en archivos secundarios del tipo <module>_projects.txt.

Expansión por módulo y por proyecto. Para cada proyecto listado:

checkout-projects determina la ruta efectiva según packing.svn.branch,

enruta dinámicamente a checkout-projects-branches, checkout-projects-tags o checkout-projects-trunk,

intenta ejecutar svn cleanup si existe una working copy previa,

si persiste el riesgo de bloqueo o corrupción, elimina el directorio local,

ejecuta un checkout limpio y no interactivo mediante SVN y SSH.

Cierre del paso y evidencia de finalización.

Se registra paso4.end.time y se emite la confirmación de que el PASO 4 terminó correctamente, dejando listo el resumen de tiempos global del pipeline.

Flujo por subtargets auxiliares

1. repository-projects.

Recibe moduleName y lanza buildAll sobre el archivo <moduleName>_projects.txt usando el target checkout-projects.

2. checkout-projects.

Recibe projectName y redirige a checkout-projects-branches, checkout-projects-tags o checkout-projects-trunk según el valor de packing.svn.branch.

3. checkout-projects-branches, checkout-projects-tags y checkout-projects-trunk.

Implementan la misma política de robustez:

1. detectar una working copy preexistente,
2. ejecutar svn cleanup para remover bloqueos residuales,
3. eliminar la carpeta si persiste el riesgo de lock,
4. ejecutar checkout limpio con --no-auth-cache y --non-interactive.

El uso de svn cleanup es importante porque Subversion lo define como un mecanismo para limpiar working copies, remover locks y retomar operaciones incompletas. Cuando una working copy está en un estado inconsistente, la práctica recomendada es limpiar o rehacer el checkout para recuperar una copia válida.

Resultado técnico del paso

Al finalizar correctamente este paso, deben cumplirse estas condiciones:

1. repositoriesFile queda resuelto y consumido sin errores.
2. Todos los módulos del catálogo maestro fueron iterados.
3. Todos los proyectos de cada módulo quedaron descargados en el workspace.
4. No quedan locks SVN bloqueando el ciclo de build posterior.
5. paso4.start.time y paso4.end.time quedan registrados para trazabilidad.

Entradas

1. conf/int/repositories.txt.
2. Archivos *_projects.txt por módulo.
3. Propiedades de versión resueltas en pasos previos.

Gestión de robustez

1. Mitigación de locks SVN en working copies.
2. Reintento limpio mediante eliminación del directorio local.
3. Checkout no interactivo para evitar bloqueos por prompts en Jenkins.
4. Validación temprana del archivo maestro de repositorios.

Evidencias de log

1. Ruta de repositoriesFile usada.
2. Mensajes por módulo y proyecto durante el checkout.
3. Confirmación de finalización del PASO 4.
4. Mensajes de cleanup o eliminación cuando existía una working copy previa.
5. Mensajes de checkout desde branch, tag o trunk según la ruta efectiva.

11.3.7 PASO 5 – Obtención de la última versión liberada

Este paso convierte un archivo histórico de versiones en dos valores operativos que gobiernan el resto de la corrida:

1. la versión final base contra la que se comparan y preparan los scripts de actualización,
2. la firma o identificador asociado a esa versión.

En términos prácticos, es el punto donde el pipeline fija el piso de versión sobre el cual se ejecutarán los cambios en la base de datos y el despliegue de la suite. A partir de este paso, todas las referencias de versión en el pipeline se alinean con el mismo valor oficial.

1. Validación del prerequisito: runtime de VEAntTask disponible.

El target verifica que el JAR de VEAntTask y su classpath asociado estén disponibles. Esto es obligatorio, ya que la lógica de cálculo no es nativa de Ant, sino que está encapsulada en la clase ve.task.LastFinalVersion.

2. Registro de la task LastFinalVersion.

Se declara taskdef name="lastVersion" apuntando a ve.task.LastFinalVersion con classpathref vetask. Desde ese momento, Ant puede ejecutar esta tarea como una instrucción más del flujo.

3. Lectura de la fuente de verdad de versiones.

La tarea LastFinalVersion consume el archivo definido en versions.file, normalmente conf/versions.txt. Este archivo contiene el historial de versiones finales liberadas y permite identificar cuál es la última versión confiable para el despliegue.

4. Resolución de propiedades de salida.

LastFinalVersion calcula y deja en la memoria de Ant dos propiedades clave:

- last.final.version
- last.final.version.sig

5. Estas propiedades quedan disponibles inmediatamente para los pasos que preparan scripts SQL, reemplazos de tokens y lógica de actualización.

6. Persistencia de resultados en configuración global.

El target escribe ambos valores en conf/global.properties usando la tarea propertyfile. Esto permite:

- mantener trazabilidad explícita de la versión usada en cada corrida,
- que otros targets o sub-builds lean exactamente la misma referencia de versión.

7. Evidencia en consola.

El paso imprime la versión encontrada y confirma la escritura en global.properties. Estas líneas constituyen la evidencia mínima para auditoría y diagnóstico de incoherencias de versión.

Resultado técnico del paso

Al finalizar correctamente este paso, deben cumplirse estas condiciones:

1. last.final.version definida en la sesión de Ant.
2. last.final.version.sig definida en la sesión de Ant.

3. Ambas propiedades persistidas en conf/global.properties.
4. Valores visibles en el log para trazabilidad.
5. Paso 6 y la fase de actualización de base de datos consumiendo la misma referencia de versión.

Entradas

- versions.file (normalmente conf/versions.txt).
- Runtime de VEAntTask compilado en pasos previos.
- classpathref vetask correctamente resuelto.

Salidas

- Propiedad last.final.version en la sesión de Ant.
- Propiedad last.final.version.sig en la sesión de Ant.
- Persistencia de ambas propiedades en conf/global.properties.

Impacto funcional

1. Alimenta tokens de scripts SQL de versionamiento.
2. Alinea el despliegue con la cadena histórica de releases.
3. Evita que el pipeline ejecute actualizaciones con una referencia de versión ambigua.
4. Provee una base común para build-release-source y update-environment.

Evidencias de log

- Mensaje con la última versión encontrada.
- Confirmación de escritura en global.properties.

- Ausencia de errores en taskdef lastVersion.

11.3.8 PASO 6 – Generación del paquete fuente y metadatos de módulos

Este paso responde dos preguntas críticas que requiere el proceso de liberación:

1. Qué se empaqueta exactamente.

Qué revisiones de módulos están incluidas en el release actual.

2. Dónde quedó el artefacto resultante.

Dónde se ubicó el paquete fuente que será usado como base de distribución y despliegue.

De esta forma, el PASO 6 consolida tanto la trazabilidad como la construcción operativa del release, asegurando que el paquete fuente refleje fielmente el estado descargado en PASO 4 y la versión base resuelta en PASO 5.

Vista general del flujo

El paso se organiza en dos subfases:

1. Subfase 6A: metadatos de módulos (info-all-modules-standalone).

Captura una “foto” de las versiones de módulos críticos antes del empaquetado.

2. Subfase 6B: construcción del release source (build-release-source-standalone).

Genera la estructura de distribución fuente que servirá como base para despliegues posteriores.

6A) Metadatos de módulos

Objetivo de 6A

Registrar información SVN detallada de módulos clave para que el release quede asociado a revisiones concretas. Esta trazabilidad es fundamental para auditoría y diagnóstico de diferencias entre versiones.

Módulos consultados

Los módulos típicamente evaluados incluyen, entre otros:

- Utils
- Agent
- Templates
- Scripts
- WebSite
- VEFile

Cómo funciona internamente

1. Se registra la tarea personalizada svesvn usando classpathref vetask.
2. Se ejecuta el comando info sobre la ruta de cada módulo en el workspace.
3. Se extraen y publican propiedades por módulo (basedir y version).
4. Se imprimen en consola los resúmenes de cada módulo procesado.

Salida de 6A

- Evidencia en consola de basedir y version para cada módulo crítico.
- Propiedades temporales en la memoria de Ant que permiten trazabilidad durante la corrida.

6B) Build release source

Objetivo de 6B

Generar la estructura de distribución fuente sobre la que se apoyan tanto el despliegue por ambientes como el resto del pipeline.

Cómo funciona internamente

1. Se limpia output.dir para eliminar residuos de corridas anteriores.
2. Se recrea output.dir desde cero, dejando un estado determinista.
3. Se invoca build_src.xml con el target build_dist_src.
4. Se construye la estructura dist y los artefactos de release source.

Garantías de 6B

- No hay mezcla de artefactos antiguos con la corrida actual.
- El paquete fuente refleja exactamente el estado descargado en PASO 4 y la versión base resuelta en PASO 5.

Flujo interno detallado

1. Dependencias previas.

Este paso depende de get-last-version-standalone e info-all-modules-standalone, por lo que entra con versión base resuelta y metadatos de módulos disponibles.

2. Higiene de salida.

Se ejecuta una operación de delete sobre el directorio de salida, seguida de un mkdir. Esto elimina residuos de corridas anteriores y reduce errores de empaquetado inconsistentes.

3. Construcción por sub-build.

El target llama a build_src.xml para ejecutar build_dist_src, separando la orquestación (en build_deploy.xml) de la lógica detallada de empaquetado fuente.

4. Publicación de estado.

Se emiten mensajes de éxito para el directorio de salida y la generación del paquete fuente, usados como punto de control para la continuidad hacia despliegue.

Resultado técnico del PASO 6

Al finalizar correctamente este paso, deben cumplirse estas condiciones:

1. Metadatos SVN de módulos críticos registrados en consola.
2. output.dir recreado limpio en la corrida actual.
3. Estructura de release source generada por build_dist_src.
4. Artefactos de distribución disponibles para despliegue posterior.
5. Trazabilidad suficiente para auditoría del contenido del release.

Entradas

- Módulos descargados en PASO 4.
- Versiones base resueltas en PASO 5.
- Rutas y archivos de build definidos en global.properties.
- Runtime de VEAntTask para consultas SVN en la subfase 6A.

Salidas

- Evidencia de versiones de módulos (6A).
- Directorio dist regenerado (6B).
- Paquete fuente y estructura de release listos para despliegue.

Evidencias de log

- Confirmación de metadatos por módulo.
- Confirmación de creación/limpieza de output.dir.
- Confirmación de ejecución de build_dist_src.
- Confirmación final de paquete fuente generado.

11.3.9 PASO 7 – Despliegue a entornos (loop multi-entorno)

Este paso transforma el release ya preparado en una ejecución real por ambientes. Su función principal es aplicar la misma lógica de actualización de forma repetible para cada entorno objetivo, manteniendo aislamiento de propiedades y trazabilidad por iteración.

En términos prácticos, PASO 7 es el puente entre el build de artefactos y la actualización efectiva de plataformas de prueba y soporte: el pipeline deja de trabajar con definiciones abstractas y pasa a ejecutar operaciones concretas sobre servidores, bases de datos y sitios web.

Flujo interno

1. deploy-to-environments-standalone – validación de artefacto y arranque del flujo

Este subtarget es el punto de entrada al bloque de despliegue.

1. Validación del artefacto de release.

Antes de iterar ambientes, el target verifica si existe dist/release-<version>.zip.

- Si el archivo existe, se usa directamente, evitando una reconstrucción innecesaria y optimizando el tiempo de ejecución.

- Si no existe, se invoca build-release-source-standalone para garantizar que hay un paquete base disponible.

2. Carga de propiedades de integrador.

Se carga el archivo custom_integrator.properties para obtener parámetros globales de despliegue, como cantidad de ambientes y configuraciones de conectividad.

- Si el archivo no existe, se emite un warning explícito en el log para facilitar el diagnóstico.

3. Control temporal de ejecución.

Se registra paso7.start.time para medir la duración total del paso. Dentro del bloque de actualización (7B) también se registran paso7b.start.time y paso7b.end.time, lo que permite discriminar entre el tiempo de preparación y el tiempo de ejecución real de despliegue.

2. update-test-environment-standalone – inicialización del loop multi-entorno

Este subtarget coordina la iteración sobre los ambientes.

1. Lectura del número de entornos.

Se lee updated.environments.amount y se prepara una iteración controlada usando la tarea ve.task.TargetLoop.

- La iteración se ejecuta en secuencia (ambiente 1, luego 2, etc.), lo que reduce la concurrencia no controlada sobre recursos compartidos, como bases de datos o servidores de Tomcat.

2. Entrada al bloque de actualización por ambiente.

Por cada valor del contador, se invoca update-single-environment-standalone, pasando el índice del ambiente y las propiedades asociadas.

3. update-single-environment-standalone – procesamiento unitario por ambiente

Por cada iteración, el pipeline ejecuta un ciclo completo de actualización sobre un único ambiente, garantizando homogeneidad y aislación de errores.

1. Incremento del contador interno.

Se actualiza una variable que identifica el ambiente actual dentro de la corrida.

2. Validación de propiedades obligatorias.

Antes de continuar, se verifica que las propiedades esenciales para el ambiente estén presentes.

3. Mapeo de propiedades indexadas a propiedades activas.

Las propiedades indexadas upd.environmentN.* se copian a un conjunto de propiedades de uso general, como upd.environment.*, tomcat.*, site.title y environment.tomcat.*. Este mapeo evita duplicar lógica por índice y permite que los mismos targets internos trabajen con un conjunto homogéneo de variables, independientemente de cuál sea el número del ambiente.

4. Ejecución del ciclo completo de actualización.

Para cada ambiente, el flujo ejecuta, de forma coordinada, las siguientes acciones:

1. validación de configuración mínima del ambiente,
2. undeploy del sitio actual,
3. ejecución de stop de Tomcat por SSH,
4. backup pre-actualización de la base de datos,
5. ejecución de script de versión en la base de datos,
6. actualización de la base de datos con VeUpdateSuite,
7. reinicio de Tomcat,
8. despliegue del sitio actualizado.

5. Este orden está diseñado para minimizar el riesgo de inconsistencia entre la aplicación y la base de datos durante la ventana de actualización.

6. Registro de resultado por ambiente.

Tras concluir la ejecución, se imprime un mensaje de éxito o fallo asociado al ambiente actual, lo que permite seguir el progreso del loop.

4. Cierre del bloque 7B y del PASO 7

Al finalizar todas las iteraciones:

- se registran los tiempos de cierre del bloque 7B y del PASO 7 completo,
- se emite un mensaje de confirmación de que el paso de despliegue ha terminado,
- el pipeline queda listo para la fase de validación y resumen final.

Flujo de decisión clave en PASO 7

El paso sigue un pequeño árbol de decisiones para adaptarse a distintas condiciones:

1. Si el artefacto de release existe:

el pipeline continúa directamente hacia el despliegue en los ambientes, saltándose la reconstrucción.

2. Si el artefacto de release no existe:

el sistema reconstruye primero el release mediante build-release-source-standalone y luego procede al despliegue.

3. Si falta una propiedad obligatoria en un ambiente:

la iteración correspondiente falla con un fail inmediato, evitando actualizaciones parciales o silenciosas que podrían dejar el sistema en un estado inconsistente.

Resultado técnico del PASO 7

Al finalizar correctamente este paso, deben cumplirse estas condiciones:

1. El subpaso 7B se ejecuta para todos los ambientes definidos en updated.environments.amount.
2. Cada iteración valida las propiedades del ambiente antes de actualizar.
3. Se registran paso7.start.time y paso7.end.time, así como los tiempos específicos del bloque 7B.
4. Cada ambiente deja un registro explícito de éxito o fallo en el log.
5. El pipeline queda listo para el resumen final, sin tareas pendientes de despliegue.

Entradas

1. Release source disponible en dist (o capacidad de reconstruirlo).
2. custom_integrator.properties con parámetros globales.
3. updated.environments.amount.
4. Propiedades por ambiente con el patrón upd.environmentN.*.

Salidas

1. Ejecución del ciclo de actualización sobre N ambientes.
2. Evidencia de tiempo total y subtiempo del bloque 7B.
3. Registro por ambiente de éxito o fallo durante la iteración.

Contrato obligatorio por ambiente

Se exige que cada ambiente esté correctamente definido mediante las siguientes propiedades:

1. upd.environmentN.tomcat.url
2. upd.environmentN.tomcat.user
3. upd.environmentN.tomcat.pwd

4. upd.environmentN.prefix
5. upd.environmentN.dao.db
6. upd.environmentN.jdbc.URL
7. upd.environmentN.jdbc.user
8. upd.environmentN.jdbc.pwd

Si alguna de estas propiedades no está presente, el proceso termina con un fallo inmediato en la iteración correspondiente, evitando despliegues parciales sin control.

Mapeo y normalización de propiedades

Antes de actualizar cada ambiente, se realizan las siguientes transformaciones de propiedades:

1. upd.environmentN.* → upd.environment.* para uso generalizado en la ejecución.
2. valores de tomcat.* y site.title para los targets de despliegue.
3. environment.tomcat.* para las operaciones de stop/start por SSH.

Este diseño desacopla la lógica de despliegue de la numeración de ambientes, permitiendo que el mismo conjunto de targets internos funcione indistintamente para el ambiente 1, 2 o N.

Evidencias de log

Inicio de PASO 7 y del bloque 7B, con timestamps.

Número de ambientes a procesar, extraído de updated.environments.amount.

Confirmación de cada ambiente completado, identificando explícitamente el índice.

Resumen final de tiempos de despliegue.

Mensajes de fallback o warning cuando falta custom_integrator.properties.

Mensajes por iteración que identifican el ambiente actual y el estado de la operación.

11.4 Flujo interno de actualización de un entorno específico

El PASO 7 representa la fase crítica de despliegue multi-entorno, donde el pipeline aplica el paquete fuente generado a cada cliente configurado de forma secuencial y autónoma. Este paso utiliza una arquitectura de loop parametrizado que itera desde 1 hasta updated.environments.amount, invocando para cada entorno (upd.environmentX.*) un flujo estandarizado de 7 sub-pasos que garantiza atomicidad y recuperación ante fallos individuales sin afectar otros entornos (RF9, RNF3).

11.4.1 Validaciones iniciales y normalización de propiedades

En la actualización de un entorno se incrementa un índice interno y se construyen nombres de propiedades dinámicas (por ejemplo, upd.environment1.tomcat.url, upd.environment2.tomcat.url), que se validan como obligatorias; si falta alguna, la ejecución se detiene con un mensaje explícito indicando cuál propiedad no fue encontrada.

Luego se copian estas propiedades específicas a un conjunto genérico (upd.environment.tomcat.url, upd.environment.jdbc.URL, etc.) y a propiedades de despliegue esperadas por otros targets (tomcat.url, site.title, etc.), lo que permite reutilizar la misma lógica sin duplicar código para cada entorno.

También se procesa la propiedad que indica el tipo de base de datos (dao.db) para extraer el motor (Oracle o SQL Server.), guardándolo en una propiedad engine que se utilizará en la ejecución de scripts y en la configuración del dialecto.

11.4.2 Undeploy del sitio web

Antes de actualizar, se ejecuta un undeploy del sitio existente en el Tomcat del entorno, detectando la versión de Tomcat (6, 7, 8, 9) para elegir la URL correcta del administrador (/manager o /managerSVE).

El undeploy se hace mediante peticiones autenticadas al manager utilizando tomcat.user y tomcat.pwd, y su resultado se guarda en un archivo temporal de log, pero sin detener el proceso si el undeploy encuentra el sitio ya inexistente (para evitar fallos innecesarios).

Una vez completado el undeploy, se informa en consola que el sitio ha sido retirado del servidor de aplicaciones, dejando el entorno listo para detener el servicio y aplicar cambios en la base de datos y en los archivos.

11.4.3 Detención controlada de Tomcat por SSH

El pipeline detiene el servicio Tomcat conectándose por SSH al host configurado (environment.tomcat.host) usando credenciales de contraseña o bien un archivo de clave privada con frase de paso, según lo que se haya definido en las propiedades del entorno.

El comando remoto ejecutado es el script de parada (environment.tomcat.host.stop.shell), que típicamente invoca el shutdown.sh o el script equivalente en el sistema operativo configurado.

Después de ejecutar el comando de parada, se introduce una espera (por ejemplo, 10 segundos) por medio de comandos remotos (sleep o Start-Sleep) para asegurar que el servicio se detiene por completo antes de continuar con las tareas sobre archivos y base de datos.

11.4.4 Ejecución respaldo de base de datos

El pipeline invoca un proceso de backup usando la configuración del entorno (host Oracle, usuario, contraseña y directorio de exportación), típicamente apoyándose en el comando exp configurado en las propiedades del entorno.

Para cada entorno se usan parámetros como:

Host de base de datos Oracle (por ejemplo temporal.pensemos.cloud o uai.pensemos.cloud)

Usuario de exportación (suiteve / manuales u otro técnico)

Contraseña (upd.environmentX.db.sys.pwd / db.system.pwd)

Servicio/local service (xe u otro)

Directorio de backup (por ejemplo sve/exports en el servidor)

El resultado esperado es un archivo de exportación (dump) generado en el directorio configurado, dejando una copia del estado de la base de datos antes de aplicar cambios de versión o de estructura.

Si el backup falla (problema de credenciales, espacio o comando no disponible), el pipeline debe detenerse y no continuar con los pasos de script de versión ni actualización, para evitar dejar la base de datos en un estado parcialmente modificado sin respaldo reciente.

11.4.5 Ejecución del script de versión en base de datos

Para registrar la versión desplegada, se copia y adapta un script de versión a un archivo temporal, reemplazando marcadores por la versión y la firma (last.final.version, last.final.version.sig) obtenidas en el PASO 5.

Se ajusta la URL JDBC según el motor de base de datos: para SQL Server se agregan parámetros como trustServerCertificate, loginTimeout, socketTimeout y encrypt, mientras que para Oracle se utiliza una URL estándar de tipo jdbc:oracle:thin:@host:puerto:sid.

Finalmente se ejecuta el script SQL utilizando el driver correcto (por ejemplo, ojdbc.jar o sqljdbc.jar), las credenciales configuradas del entorno y un classpath extendido, registrando en consola el resultado de la operación.

11.4.6 Actualización de base de datos con VeUpdateSuite

Una vez actualizada la versión, se ejecutan los procesos de actualización de la base de datos de la Suite usando la lógica de VeUpdateSuite, que se parametriza con el motor, la URL JDBC, usuario, contraseña y propiedades específicas para el tipo de base.

La actualización recorre scripts que modifican estructura y datos, asegurando que el esquema de la base de datos quede compatible con la nueva versión de la aplicación.

Al terminar, se ejecuta un segundo proceso de actualización de base para completar ajustes adicionales, manteniendo en todo momento los mismos parámetros de conexión y licenciamiento.

11.4.7 Arranque de Tomcat y despliegue del sitio

Con la base de datos ya actualizada, se inicia el servicio Tomcat mediante SSH, utilizando las mismas credenciales o claves que en la detención, pero esta vez ejecutando el script de arranque (environment.tomcat.host.start.shell).

De nuevo, se agrega una espera controlada para dar tiempo a que el contenedor se levante completamente antes de desplegar el sitio.

El despliegue del sitio se realiza invocando un target que utiliza la URL y credenciales del manager de Tomcat, la URL JDBC del entorno, las credenciales de base de datos y la ruta de VEFile (carpeta donde residen configuraciones, licencias y archivos de la Suite), todo derivado de las propiedades del entorno.

Al finalizar, se informa en consola que el entorno con el prefijo correspondiente (por ejemplo, suiteve o manuales) ha sido actualizado exitosamente.

Figura 3

Mensaje sitio desplegado

```
build-and-deploy-standalone:
     [echo] ========================================
     [echo] Iniciando Build and Deploy Standalone
     [echo] Hora de inicio: 2026-03-04 10:15:55
     [echo] ========================================
     [echo] ✓ Todos los pasos completados:
     [echo]   1. ✓ Info SVN del paquete
     [echo]   2. ✓ Checkout del VEAntTask
     [echo]   3. ✓ Compilación del VEAntTask
     [echo]   4. ✓ Checkout de todos los módulos
     [echo]   5. ✓ Obtención de última versión
     [echo]   6A. ✓ Info SVN de todos los módulos
     [echo]   6B. ✓ Generación de paquete fuente
     [echo]   7. ✓ Actualización de ambientes de prueba
     [echo]     ✓ Undeploy de sitios web
     [echo]     ✓ Detención de Tomcat
     [echo]     ✓ Ejecución de scripts SQL
     [echo]     ✓ Inicialización de tareas personalizadas
     [echo]     ✓ Información de VeUpdateSuite
     [echo]     ✓ Construcción y compilación de módulos VUS
     [echo] ✓ Build and Deploy completado exitosamente
     [echo] ========================================
     [echo] 📊 RESUMEN DE TIEMPOS DE EJECUCIÓN:
     [echo] ========================================
     [echo] 🕐 Tiempo total: 2026-03-04 10:15:55 → 2026-03-04 12:13:43
     [echo] 📋 Detalle por pasos:
     [echo]   PASO 1 (Info SVN):     2026-03-04 10:15:48 → 2026-03-04 10:15:48
     [echo]   PASO 2 (Checkout VETask): 2026-03-04 10:15:48 → 2026-03-04 10:15:53
     [echo]   PASO 3 (VETask build/info): 2026-03-04 10:15:53 → 2026-03-04 10:15:54
     [echo]   PASO 4 (Checkout módulos): 2026-03-04 10:15:55 → 2026-03-04 10:42:18
     [echo]   PASO 5 (Última versión): 2026-03-04 10:42:18 → 2026-03-04 10:42:18
     [echo]   PASO 6 (Paquete fuente / Info módulos): 2026-03-04 10:42:18 → 2026-03-04 10:52:01
     [echo]   PASO 7 (Despliegue total): 2026-03-04 10:52:01 → 2026-03-04 12:13:43
     [echo] ========================================
```

11.5 Prerrequisitos para ejecución del pipeline

Antes de ejecutar el pipeline Build and Deploy Standalone de la Suite Visión Empresarial, es fundamental que el entorno de Jenkins cumpla con los requisitos mínimos de hardware, software, conectividad y plugins. Estos prerrequisitos garantizan que el proceso se complete sin interrupciones técnicas, especialmente considerando el volumen de datos que maneja (checkout de múltiples módulos, compilación de VEAntTask, generación de WARs de hasta 1.5 GB y despliegues multi-entorno).

11.5.1 Requisitos de hardware de la máquina Jenkins

Espacio en disco: Mínimo 10 GB libres en el volumen donde reside el workspace de Jenkins. Este espacio se consume principalmente por:

Checkout del empaquetado SVN (~100-500 MB)

Checkout de VEAntTask y todos los módulos (~2-4 GB)

Directorios temporales de compilación y distribución (~3-5 GB)

Artefactos WAR generados por sitio (~1.5 GB cada uno en configuraciones completas)

Logs detallados y copias de seguridad de base de datos durante despliegues

CPU: Recomendado 4 núcleos físicos (8 hilos) mínimo para una ejecución aceptable. La compilación paralela de módulos Java y las operaciones SSH simultáneas a múltiples entornos aprovechan múltiples núcleos; con menos de 4 núcleos, los tiempos de compilación (PASO 3 y PASO 4) se extienden significativamente.

RAM: Mínimo 8 GB asignados al proceso de Jenkins. La compilación de la Suite consume memoria intensivamente durante:

Compilación de 57+ clases de VEAntTask (PASO 3)

Checkout y compilación paralela de 20+ módulos (PASO 4)

Generación de paquetes WAR grandes (PASO 6 y PASO 7) Máximo recomendado: 16 GB para despliegues con 3+ entornos simultáneos.

Nota: Para despliegues concurrentes o builds frecuentes, se recomienda una máquina dedicada con SSD, 8 núcleos y 16-32 GB RAM para evitar cuellos de botella y permitir escalabilidad horizontal.

11.5.2 Conectividad de red y acceso VPN

Acceso VPN corporativo obligatorio: La máquina Jenkins debe estar conectada permanentemente a la VPN interna de PENSEMOS S.A., ya que todos los repositorios SVN residen en infraestructura protegida (svn.pensemos.cloud).

Sin VPN activa, el PASO 0 (checkout SVN del empaquetado) y PASO 2 (checkout VEAntTask) fallará inmediatamente con errores de autenticación o "host unreachable".

Los despliegues por SSH (PASO 7) hacia entornos como temporal.pensemos.cloud y uai.pensemos.cloud también requieren conectividad VPN para resolver nombres de host y acceder a puertos privados (SSH:22, Oracle:1521).

Puertos requeridos:

Tabla 11

Conectividad de red y acceso VPN.

Puerto	Protocolo	Uso
22	SSH	Stop/start Tomcat en entornos remoto

1521	TCP	Conexión JDBC Oracle XE durante actualización BD
8080/80	HTTP/HTTPS	Manager Tomcat para undeploy/deploy
3690	SVN	Checkout repositorios SVN

Latencia recomendada: < 100ms hacia servidores SVN y entornos destino para evitar timeouts en operaciones largas (checkout grandes, SSH batch).

11.5.3 Plugins obligatorios de Jenkins

Los siguientes plugins deben estar instalados y actualizados en la instancia de Jenkins:

Tabla 12

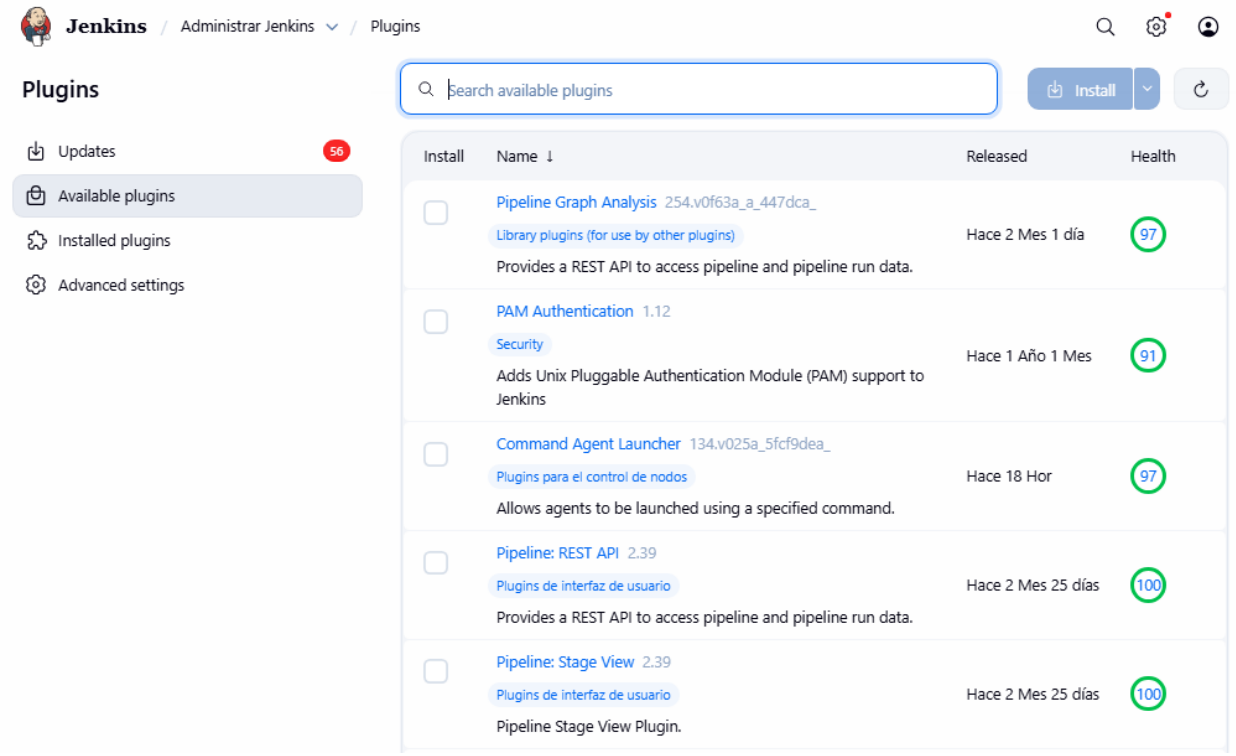
Plugins obligatorios de Jenkins

Plugin	Versión mínima	Propósito	URL
Ant Plugin	5.2.0 (vd082ecfb_16a_9)	Ejecuta el build_deploy.xml y orquesta todos los targets Ant del pipeline	plugins.jenkins.io/ant
Subversion Plugin	1.292 (ve8cf25770ee3)	Checkout inicial del empaquetado (PASO 0) y manejo nativo de SVN en Jenkins	plugins.jenkins.io/subversion
Oracle JDK Installer	8.3 (v417146707a_3d)	Instalación/management automático de JDK requerido por la Suite	plugins.jenkins.io/jdk-tool

Verificación: En Manage Jenkins > Manage Plugins > Installed, buscar estos plugins y confirmar versión ≥ mínima especificada. Si faltan, instalar desde Available Plugins.

Figura 4

Verificación de plugins



11.5.4 Entorno Java requerido

JDK 1.8.0_341 (o superior dentro de la serie 1.8): Versión soportada actualmente por la Suite Visión Empresarial y VEAntTask.

Configurar en Manage Jenkins > Global Tool Configuration > JDK Installations.

Usar el plugin Oracle JDK Installer para descarga automática desde Oracle.

Nombre de herramienta en jobs: JDK 1.8.0_341 (exacto, sensible a mayúsculas).

Configuración en job:
JDK: JDK 1.8.0_341 [To be used for this project]
JAVA_OPTS: -Xmx4096m -XX:MaxPermSize=1024m
(según capacidades)

Nota evolutiva: Cuando se migre a Java 11/17, actualizar la versión JDK en esta configuración y recompilar VEAntTask/módulos contra la nueva versión.

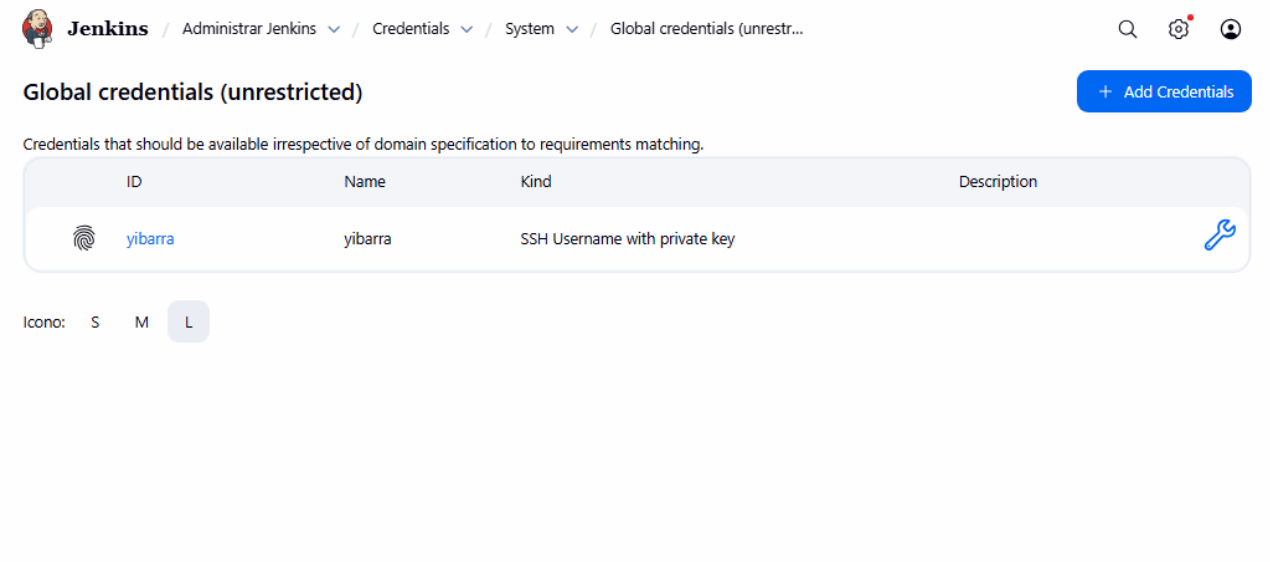
11.5.5 Credenciales y claves SSH

Credenciales SVN: Usuario yibarra (o equivalente) con acceso de lectura a todos los repositorios (/svn/ve40/branches, /svn/common, /svn/ve40/tags) ya que son los repositorios donde se encuentran los empaquetados para hacer la actualización.

Configurar en Credentials > System > Global credentials > Add > SSH Username with private key.

Figura 5

Credencial svn



Archivo clave: /var/lib/jenkins/instalaciones/svn/id_rsa (ruta absoluta referenciada en propiedades).

Credenciales por entorno (definidas en propiedades upd.environmentX.*):

Tomcat Manager: usuarios/passwords por sitio con rol manager-script, ya que es el rol que se necesita para hacer el deploy del war

SSH hosts: claves o passwords por entorno, con permisos necesarios en las rutas que se usarán (encender y apagar tomcat)

Oracle DB: usuarios/passwords por instancia

11.5.6 Verificación pre-ejecución

Antes de disparar el pipeline, ejecutar checklist:

text

[ ] 10+ GB libres en disco workspace

[ ] Jenkins conectado a VPN PENSEMOS S.A.

[ ] Plugins Ant/Subversion/JDK instalados y actualizados

[ ] JDK 1.8.0_341 configurado y seleccionado en job

[ ] Credenciales SVN configuradas (yibarra + id_rsa)

[ ] Propiedades customintegrator.properties completas (2+ entornos)

[ ] updated.environments.amount=N configurado correctamente

[ ] mark.version.from apunta a rama estable

Si algún prerrequisito falla:

- Sin SVN/VPN: PASO 0 checkout fallará inmediatamente
- Sin Ant plugin: No se ejecutará build_deploy.xml
- Sin JDK 1.8: Compilación VEAntTask fallará (PASO 3)

- Sin espacio disco: PASO 4 (checkout módulos) o PASO 6 (dist) fallará por "No space left on device"

11.6 Propiedades requeridas para el pipeline Ant

El pipeline Build and Deploy Standalone requiere un conjunto específico de propiedades que se pasan como parámetros -D desde Jenkins al ejecutar ant -file build_deploy.xml build-and-deploy-standalone. Estas propiedades se agrupan en tres categorías principales: control del pipeline, configuración multi-entorno y SVN/marcación. Todas deben definirse en el Job de Jenkins.

11.6.1 Propiedades de control del pipeline (globales)

Estas propiedades definen el comportamiento general del pipeline y son obligatorias para cualquier ejecución.

Tabla 13

Propiedades de control del pipeline (globales)

Propiedad	Valor ejemplo	Obligatoria	Descripción
updated.environments.amount	2	Sí	**Cantidad de entornos** a actualizar. Define cuántas veces iterará el loop del PASO 7 (1=un entorno, 2=dos entornos, etc.). Sin esta propiedad, el PASO 7B fallará
svn.repository.user	Yibarra	Sí	Usuario SVN para checkout de VEAntTask y módulos (PASO 2 y 4). Usado en URLs como svn+ssh://yibarra@svn.pensemos.cloud

svn.private.key.file	/var/lib/jenkins/instalaciones/svn/id_rsa	Sí	Ruta absoluta al archivo de clave privada SSH para autenticación SVN. Usado por SVNKit en todos los checkouts

11.6.2 Propiedades por entorno (upd.environmentX.)*

Obligatorias (validadas estrictamente en update-single-environment-standalone). El sufijo X va de 1 hasta updated.environments.amount. Si falta cualquiera, el pipeline falla con mensaje explícito: "Falta la propiedad obligatoria upd.environmentX.propiedad".

11.6.3 Configuración Tomcat Manager (acceso web)

Tabla 14

Configuración Tomcat Manager (acceso web)

Propiedad	Ejemplo Env1	Ejemplo Env2	Uso
upd.environmentX.tomcat.url	https://temporal.pensemos.cloud	https://uai.pensemos.cloud	**URL base del Tomcat** para manager (undeploy/deploy). Usada en PASO 7 para construir /managerSVE/text/deploy y /undeploy
upd.environmentX.tomcat.user	AdmO	Adm	Usuario del Tomcat Manager (HTTP auth)
upd.environmentX.tomcat.pwd	pass1	pass2	Contraseña del Tomcat Manager
upd.environmentX.prefix	Suiteve	manuales	**Prefijo del sitio web** (/suiteve o /manuales). Define el context path del WAR en Tomca

upd.environmentX.tomcat.version	8	8	Versión Tomcat (6/7/8/9). Determina URL manager (/manager vs /managerSVE)

11.6.4 Configuración SSH (gestión remota Tomcat)

Tabla 15

Configuración SSH (gestión remota Tomcat)

Propiedad	Ejemplo Env1	Ejemplo Env2	Uso
upd.environmentX.tomcat.host	temporal.pensemos.cloud	uai.pensemos.cloud	Host SSH donde está instalado Tomcat.
upd.environmentX.tomcat.host.user	Yibarra	yibarra	Usuario SSH para ejecutar comandos start/stop.
upd.environmentX.tomcat.host.pwd	(vacío)	pass	Contraseña SSH (si no se usa keyfile). Prioridad sobre clave privada.
upd.environmentX.tomcat.host.keyfile	/data/ssh/privada-yibarra	(vacío)	Ruta absoluta a clave privada SSH (fallback si no hay pwd).
upd.environmentX.tomcat.host.passphrase	(vacío)	(vacío)	Frase de paso de la clave privada (si aplica)
upd.environmentX.tomcat.host.so	Linux	linux	SO del host (linux/windows). Afecta comandos de espera (sleep vs Start-Sleep)

upd.environmentX.tomcat.host.start.shell	sh /home/opc/apache-tomcat-9.0.109/bin/startup.sh	sh /home/opc/apache-tomcat-9.0.82/bin/startup.sh	Comando remoto para iniciar Tomcat.
upd.environmentX.tomcat.host.stop.shell	sh /home/opc/apache-tomcat-9.0.109/bin/shutdown.sh	sh /home/opc/apache-tomcat-9.0.82/bin/shutdown.sh	Comando remoto para detener Tomcat.

11.6.5 Configuración base de datos Oracle

Tabla 16

Configuración base de datos Oracle.

Propiedad	Ejemplo Env1	Ejemplo Env2	Uso
upd.environmentX.dao.db	Oracle:XE	Oracle:XE	Tipo de base (Oracle:XE, MSSQL2008). Extrae motor para scripts
upd.environmentX.dao.engine	Oracle	Oracle	Motor de base de datos (extraído de dao.db).
upd.environmentX.db.sys.pwd	pass1	pass2	Contraseña usuario SYS de Oracle (export/backup)
upd.environmentX.db.system.pwd	pass1	pass2	Contraseña usuario SYSTEM de Oracle
upd.environmentX.jdbc.URL	jdbc:oracle:thin:@temporal.pensemos.cloud:1521:xe	jdbc:oracle:thin:@uai.pensemos.cloud:1521:xe	URL JDBC completa para scripts de versión y VeUpdateSuite
upd.environmentX.jdbc.user	Suiteve	manuales	Usuario esquema de la aplicación
upd.environmentX.jdbc.pwd	Pass	pass	Contraseña esquema aplicación
upd.environmentX.jdbc.driver	oracle.jdbc.driver.OracleDriver	oracle.jdbc.driver.OracleDriver	Driver JDBC Oracle

upd.environmentX.hibernate.dialect	org.hibernate.dialect.OracleDialect	org.hibernate.dialect.OracleDialect	Dialecto Hibernate para la aplicación

11.6.6 Configuración VEFile y despliegue

Tabla 17

Configuración VEFile y despliegue

Propiedad	Ejemplo Env1	Ejemplo Env2	Uso
upd.environmentX.lic.path	/suiteve/vefile/lic	/home/opc/vefile-manuales/lic	**Ruta absoluta licencia** en servidor destino
upd.environmentX.vefile.home	/suiteve/vefile	/home/opc/vefile-manuales	**Home VEFile** (configuraciones, licencias, archivos Suite)
upd.environmentX.tomcat.host.vefile.path	/suiteve/vefile	/home/opc/vefile-manuales	Ruta VEFile usada en SSH (debe coincidir con vefile.home)
upd.environmentX.jdk.version	8	8	Versión JDK del entorno destino

11.6.7 Manual de usuario estructura para 2 entornos a (copiar/pegar)

```
# CONTROL PIPELINE

updated.environments.amount=2

svn.repository.user=yibarra

svn.private.key.file=/var/lib/jenkins/instalaciones/svn/id_rsa


# ENTORNO 1 (Pruebas)

upd.environment1.tomcat.url=https://temporal.pensemos.cloud

upd.environment1.tomcat.user=user
```

upd.environment1.tomcat.pwd=Passwo

upd.environment1.dao.db=Oracle:XE

upd.environment1.db.sys.pwd=passorc

upd.environment1.db.system.pwd=passorc

upd.environment1.jdbc.URL=jdbc:oracle:thin:@temporal.pensemos.cloud:1521:xe

upd.environment1.lic.path=/suiteve/vefile/lic

upd.environment1.tomcat.host=temporal.pensemos.cloud

upd.environment1.tomcat.host.user=yibarra

upd.environment1.tomcat.host.keyfile=/data/ssh/privada-yibarra

upd.environment1.tomcat.host.passphrase=

upd.environment1.tomcat.host.pwd=

upd.environment1.tomcat.host.start.shell=sh /home/opc/apache-tomcat-9.0.109/bin/startup.sh

upd.environment1.tomcat.host.stop.shell=sh /home/opc/apache-tomcat-9.0.109/bin/shutdown.sh

upd.environment1.tomcat.host.so=linux

upd.environment1.tomcat.host.vefile.path=/suiteve/vefile

upd.environment1.prefix=suiteve

upd.environment1.vefile.home=/suiteve/vefile

upd.environment1.tomcat.version=8

upd.environment1.jdk.version=8

upd.environment1.dao.engine=Oracle

upd.environment1.jdbc.user=userjdbc

upd.environment1.jdbc.pwd=passorc

upd.environment1.jdbc.driver=oracle.jdbc.driver.OracleDriver

upd.environment1.hibernate.dialect=org.hibernate.dialect.OracleDialect

upd.environment1.backup.remote.dir=/home/opc/backs

upd.environment1.oracle.host.exp.command=/u01/app/oracle/product/11.2.0/xe/bin/exp

upd.environment1.oracle.host.oracle.home=/u01/app/oracle/product/11.2.0/xe

upd.environment1.oracle.host.oracle.sid=XE

ENTORNO 2 (Producción)

upd.environment2.tomcat.url=https://uai.pensemos.cloud

upd.environment2.tomcat.user=user2

upd.environment2.tomcat.pwd=pwd2

upd.environment2.dao.db=Oracle:XE

upd.environment2.db.sys.pwd=pass2

upd.environment2.db.system.pwd=pass2

upd.environment2.jdbc.URL=jdbc:oracle:thin:@uai.pensemos.cloud:1521:xe

upd.environment2.lic.path=/home/opc/vefile-manuales/lic

upd.environment2.tomcat.host=uai.pensemos.cloud

upd.environment2.tomcat.host.user=yibarra

upd.environment2.tomcat.host.keyfile=

upd.environment2.tomcat.host.pwd=passtomssh

upd.environment2.tomcat.host.start.shell=sh /home/opc/apache-tomcat-9.0.82/bin/startup.sh

upd.environment2.tomcat.host.stop.shell=sh /home/opc/apache-tomcat-9.0.82/bin/shutdown.sh

upd.environment2.tomcat.host.so=linux

upd.environment2.tomcat.host.vefile.path=/home/opc/vefile-manuales

upd.environment2.prefix=manuales

upd.environment2.vefile.home=/home/opc/vefile-manuales

upd.environment2.tomcat.host.catalinahome=/home/opc/vefile-manuales

upd.environment2.tomcat.version=8

upd.environment2.jdk.version=8

upd.environment2.dao.engine=Oracle

upd.environment2.jdbc.user=manuales

upd.environment2.jdbc.pwd=pass2

upd.environment2.jdbc.driver=oracle.jdbc.driver.OracleDriver

upd.environment2.hibernate.dialect=org.hibernate.dialect.OracleDialect

upd.environment2.backup.remote.dir=/home/opc/backs-manuales

upd.environment2.oracle.host.exp.command=/u01/app/oracle/product/11.2.0/xe/bin/exp

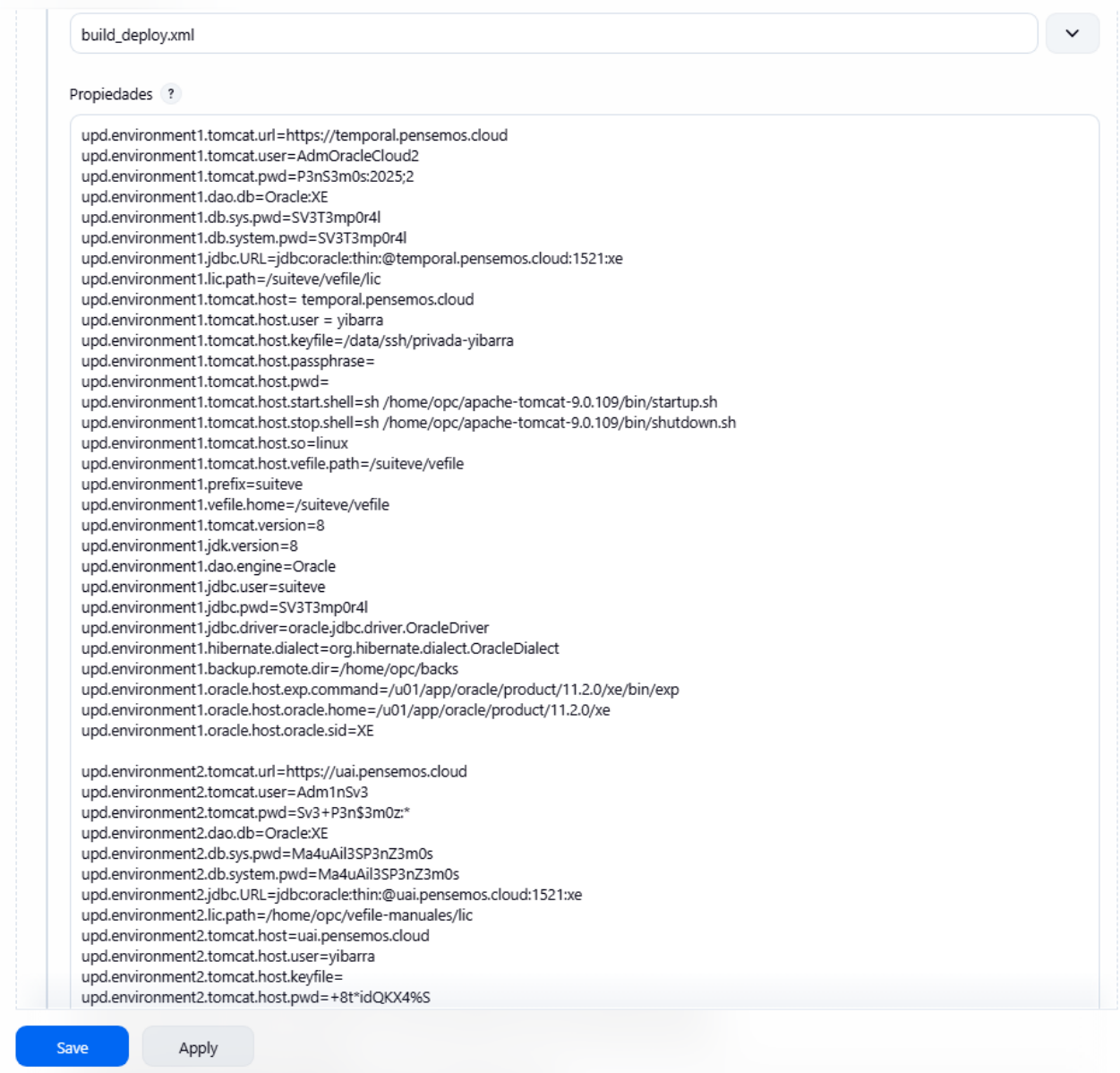
upd.environment2.oracle.host.oracle.home=/u01/app/oracle/product/11.2.0/xe

upd.environment2.oracle.host.oracle.sid=XE

Nota, se debe seguir la misma estructura para N entornos

Figura 6

Propiedades en Jenkins

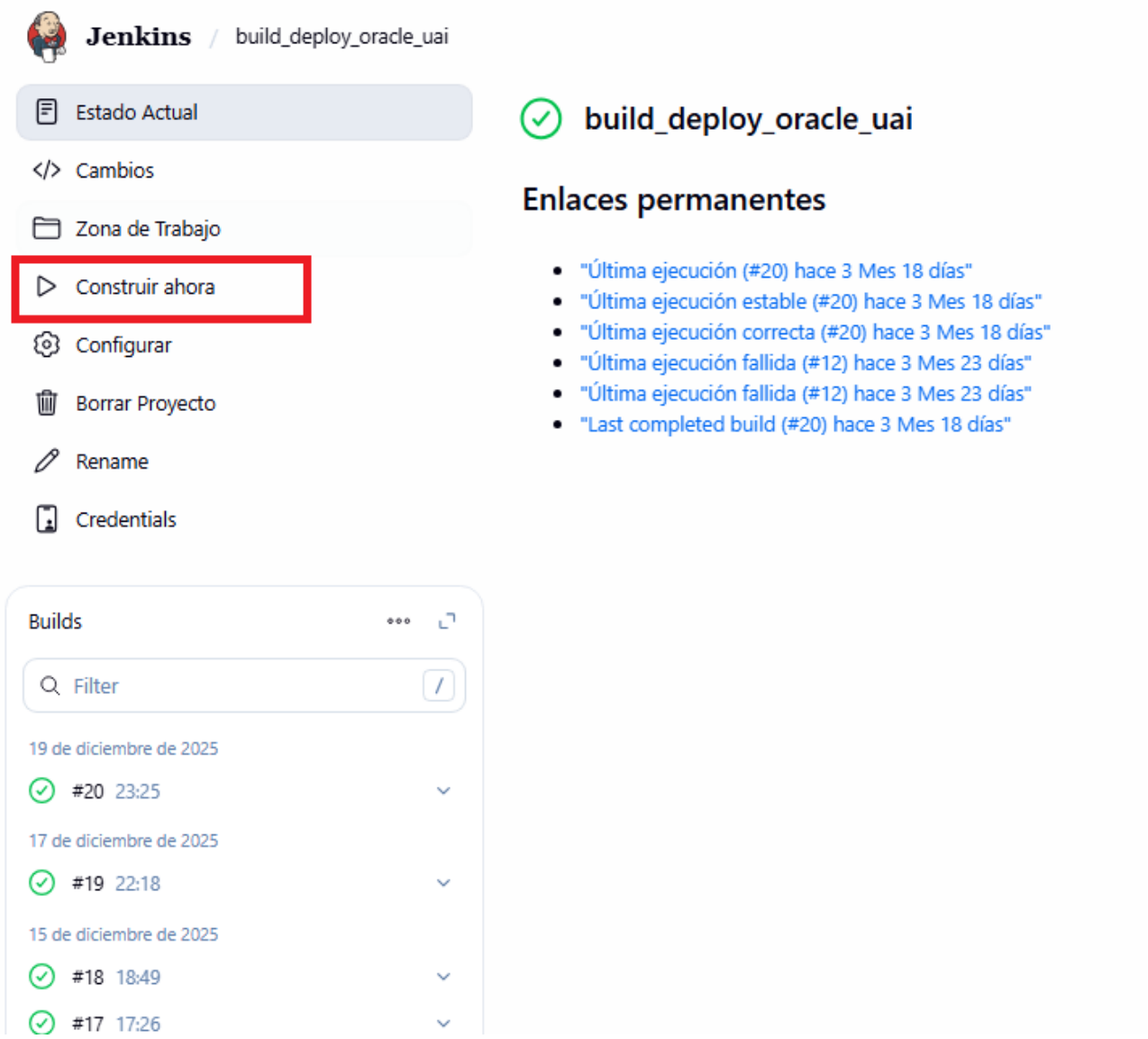


11.6.8 Lanzamiento del pipe

Una vez configurado el pipe de y Jenkins con las condiciones mencionadas se puede hacer uso del lanzamiento del pipe, yendo al pipe configurado y darle en construir ahora

Figura 7

Lanzar Ejecución



11.7 Pruebas de aceptación y validación final

Esta etapa tuvo como propósito confirmar, en un entorno de prueba provisto por PENSEMOS S.A., que el sistema de actualización masiva cumplía correctamente con el flujo esperado y que los resultados obtenidos eran consistentes tanto en Jenkins como en los entornos destino. Para ello se ejecutaron despliegues sobre ambientes como temporal.pensemos.cloud, verificando en los logs de Jenkins la finalización exitosa de cada paso, el comportamiento del pipeline y la trazabilidad completa de la actualización.

La validación final incluyó la revisión del estado de VEFile del sitio actualizado, comprobando que las rutas, licencias y archivos asociados al entorno coincidieran con la versión desplegada y con la configuración esperada para el ambiente de prueba. También se verificó que la nueva versión de la base de datos quedará aplicada correctamente, revisando tanto la ejecución de los scripts de versión como la información registrada por el proceso de actualización y respaldo.

Adicionalmente, se comprobó que la página principal del sistema permaneciera accesible desde Internet y que refleja la nueva versión desplegada, confirmando así que el WAR había sido instalado correctamente y que los módulos requeridos estaban presentes en el sitio final. Como parte de la validación operativa, se revisó que el archivo de backup generado durante el proceso se encontrará almacenado en la máquina temporal de PENSEMOS S.A., junto con la evidencia de la ejecución correspondiente, garantizando así la posibilidad de recuperación ante incidentes.

Finalmente, se confirmó que el WAR nuevo incluía los módulos necesarios para el funcionamiento esperado de la Suite Visión Empresarial, y que la actualización había sido aplicada de forma completa, sin inconsistencias visibles en la interfaz, en la base de datos ni en los registros del pipeline.

11.8 Pruebas en tiempos de ejecución

Se usó una máquina virtual para instalar Jenkins provista por PENSEMOS S.A.para realizar las pruebas, accesible en Panel de control - Jenkins necesitando la vpn de PENSEMOS S.A.para poder ingresar, con las siguientes características de sistema operativo, hardware y almacenamiento:

Tabla 18

Pruebas en tiempos de ejecución

Atributo	Valor
Propósito	Servidor Jenkins de pruebas entregado por PENSEMOS S.A.
Hostname	jenkins-dev
Tipo	Máquina virtual (QEMU / KVM)
Sistema operativo	Oracle Linux Server 9.6 (x86_64)
Kernel	Linux 6.12.0-104.43.4.2.el9uek.x86_64
Arquitectura	64 bits (x86-64)
CPU	AMD EPYC 7J13 (1 core asignado a 2 GHz, 2 hilos)
Memoria RAM	10 GiB
Disco del sistema	/dev/sda – 46 GiB (boot, EFI, LVM)
Disco de datos	/dev/sdb – 120 GiB, montado en /data (XFS)

11.8.1 Resultado de tiempos en pruebas

Tabla 19

Resultado de tiempos en pruebas

PASO	Duración aprox	Dependencias
0	3-5 min	VPN/SVN
1	<1 min	-
2	2-3 min	svn.user
3	15-25 min	JDK 1.8
4	15-20 min	10GB disco
5	<1 min	-
6	5 min	-

Nota: estos tiempos se ven estrictamente relacionados con la máquina de jenkins, la del servidor y el ancho de banda, para obtener mejores tiempos se podría tener una máquina más apropiada para este proceso.

11.9 Diagrama de flujo Build & Deploy Standalone

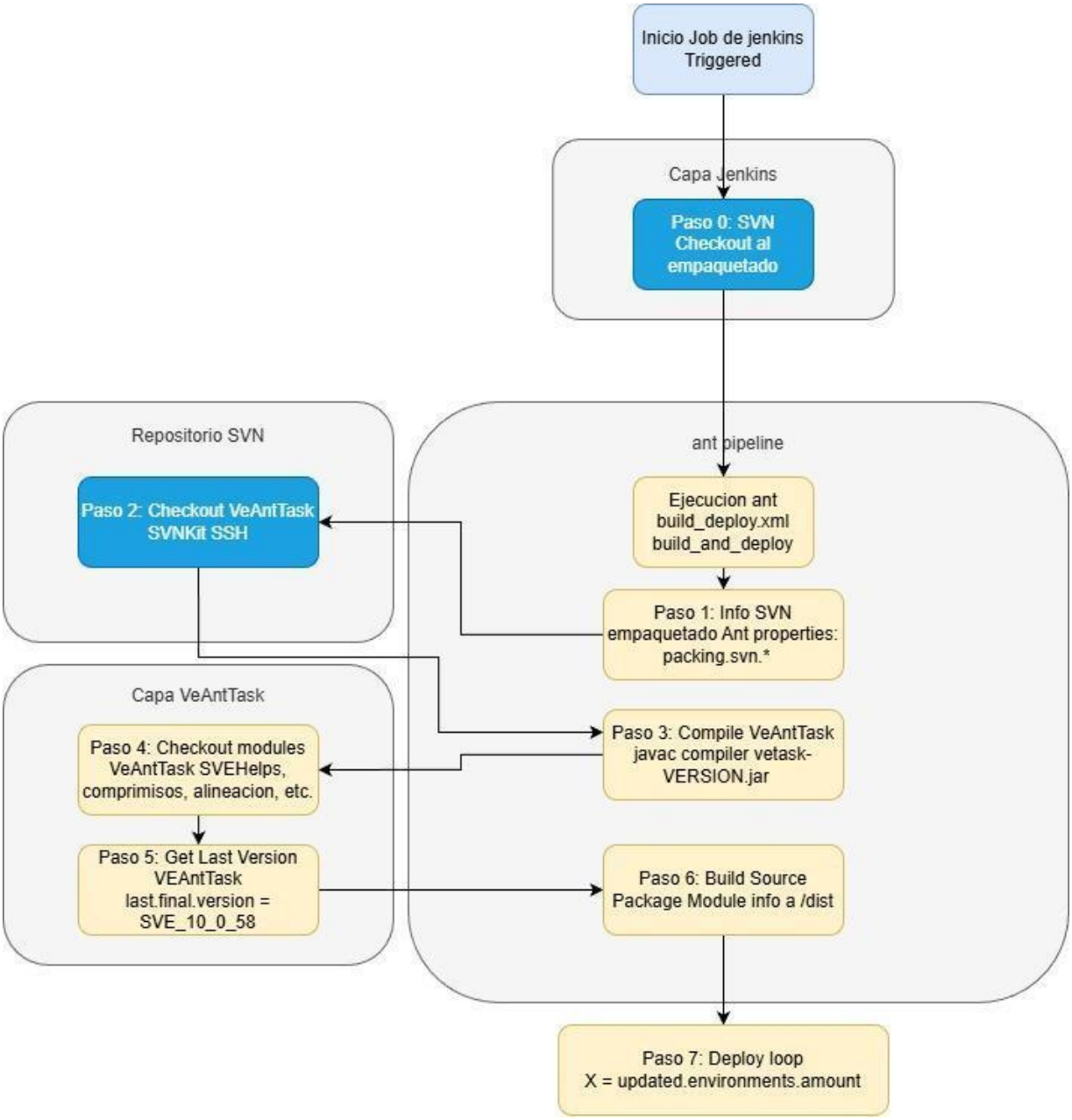
11.9.1 Diagrama de flujo del PASO 0 al 6

El primer diagrama presenta el flujo global del pipeline desde el checkout inicial del empaquetado en SVN (PASO 0) hasta la generación del paquete fuente y metadatos de módulos (PASO 6) de manera general.

Este diagrama resume, en una sola vista, cómo Jenkins coordina las tareas de obtención de código, compilación de VEAntTask, checkout de todos los módulos de la Suite y construcción del artefacto de distribución, antes de entrar en la fase de despliegue multi-entorno.

Figura 8

Flujo paso 0 a 6



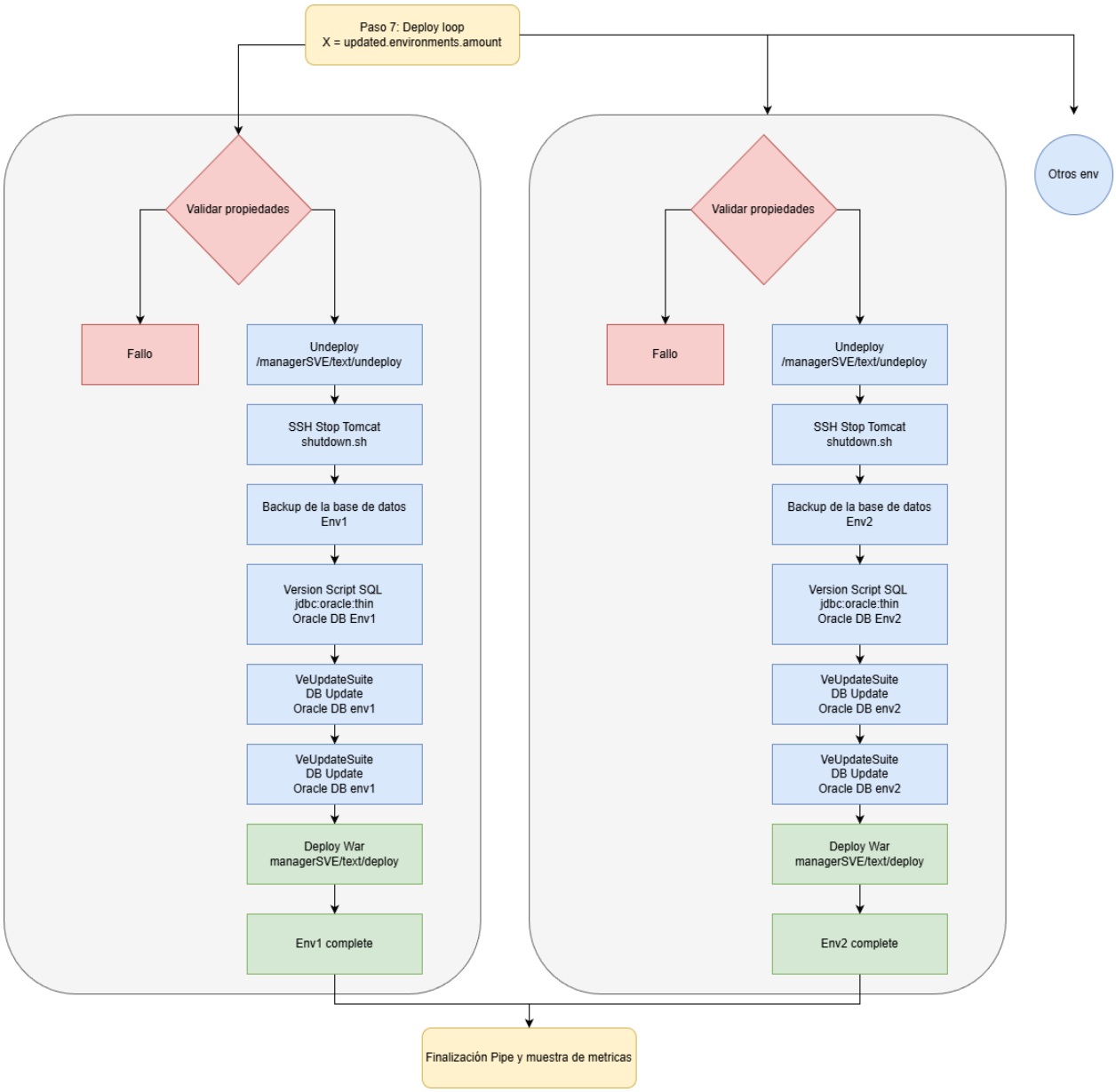
11.9.2 Diagrama de flujo paso 7

El segundo diagrama muestra el flujo cíclico interno del PASO 7, donde el pipeline recorre desde 1 hasta updated.environments.amount para actualizar cada entorno configurado.

En este esquema se visualizan los sub-pasos por entorno (validaciones, undeploy, stop de Tomcat, backup, actualización de base de datos con VeUpdateSuite, arranque y deploy del sitio), enfatizando que cada entorno sigue el mismo flujo estándar pero con propiedades propias, lo que habilita la actualización masiva controlada.

Figura 9

Flujo paso 7



12 Conclusiones

El desarrollo de este trabajo permitió diseñar e implementar un sistema de actualización masiva para la Suite Visión Empresarial (SVE) en clientes alojados en la nube, respondiendo a una necesidad operativa real de Pensemos S.A. La propuesta soluciona una problemática asociada a los procesos manuales, secuenciales y dependientes del equipo de soporte, que generaban altos tiempos de ejecución, riesgo de errores y dificultad para mantener trazabilidad en cada actualización.

La adopción de una arquitectura basada en CI/CD y principios DevOps demostró ser una alternativa adecuada para automatizar y controlar el flujo de actualización. Jenkins cumplió un papel central como orquestador del proceso, integrando herramientas ya existentes en el ecosistema de la organización, como Subversion, Apache Ant, VEAntTask, Tomcat y Oracle, lo que permitió aprovechar la infraestructura disponible sin requerir cambios disruptivos.

La estructuración del proceso en pasos definidos facilitó la construcción de un pipeline modular, trazable y mantenible. Cada fase del sistema, desde la obtención del contexto SVN hasta la ejecución de los despliegues por entorno, fue diseñada para reducir el acoplamiento entre tareas, mejorar la validación de prerrequisitos y facilitar el diagnóstico de fallos, fortaleciendo así la confiabilidad del sistema.

Asimismo, la incorporación de mecanismos de respaldo, control de errores, validación de propiedades y registro detallado en consola aportó mayor seguridad y robustez al proceso. Esto permitió que el sistema no solo realizará actualizaciones de forma automatizada, sino que también ofreciera un nivel adecuado de supervisión y recuperación ante incidencias, lo cual es esencial en entornos empresariales críticos.

La validación del sistema en entornos de prueba proporcionados por Pensemos S.A. confirmó que la solución cumple con los objetivos propuestos. Los resultados evidenciaron que es posible ejecutar actualizaciones controladas, verificar la correcta instalación de los artefactos, confirmar la actualización de la base de datos y mantener la operatividad del sitio, reduciendo significativamente la intervención manual.

Finalmente, el trabajo dejó una base sólida para futuras mejoras, tanto en la ampliación del sistema hacia el despliegue de sitios nuevos como en la incorporación de notificaciones automáticas por correo electrónico. En conjunto, la solución desarrollada contribuye al fortalecimiento de los procesos de automatización de Pensemos S.A. y representa un avance importante en la modernización de la gestión de actualizaciones de la Suite Visión Empresarial.

13 Trabajo futuro

Como líneas de trabajo futuro, se identifican dos posibles extensiones del sistema desarrollado. La primera consiste en ampliar la solución para que, además de ejecutar actualizaciones sobre sitios ya existentes, también permita el despliegue de sitios nuevos dentro de la Suite Visión Empresarial. Esta evolución requeriría ajustar la lógica actual del pipeline para contemplar escenarios de aprovisionamiento inicial, configuración automática de propiedades y despliegue completo de nuevos entornos o clientes.

La segunda línea de trabajo futuro corresponde a la incorporación de un sistema de correos electrónicos para notificación automática de los sitios actualizados. Este componente permitiría informar de manera oportuna al equipo de soporte o a los responsables del proceso sobre el resultado de cada ejecución, incluyendo éxitos, fallos y observaciones relevantes, fortaleciendo así la trazabilidad y el seguimiento operativo del sistema.

Adicionalmente, se considera como una futura mejora la adaptación del sistema al proceso de modernización tecnológica que actualmente adelanta Pensemos S.A., especialmente la migración del control de versiones desde Subversion hacia Git, con GitLab como nueva plataforma de gestión del código fuente. Una vez completado este cambio, será necesario ajustar el pipeline de Jenkins para que opere sobre la nueva estructura de repositorios, permisos y flujos de integración asociados a Git, garantizando así la compatibilidad del sistema con la infraestructura tecnológica renovada de la organización.

Referencias Bibliográficas

Atlassian. (2023). Trello guides: Help getting started with Trello. https://trello.com/guide

Atlassian. (s. f.). ¿En qué consiste la integración continua? https://www.atlassian.com/es/continuous-delivery/continuous-integration

GeeksforGeeks. (s. f.). Modelo de proceso incremental - ingeniería de software. https://www.geeksforgeeks.org/software-engineering/software-engineering-incremental-process-model/

IT User. (2020, 18 de noviembre). La nube híbrida y la plataforma OpenShift de Red Hat, aceleradores de procesos en tiempos de pandemia. https://www.ituser.es/eventos/2020/11/la-nube-hibrida-y-la-plataforma-openshift-de-red-hat-aceleradores-de-procesos-en-tiempos-de-pandemia

Jenkins. (s. f.). Jenkins. https://www.jenkins.io/

Jibble. (2025). Jibble: Time tracking & attendance software. https://www.jibble.io

Microsoft. (2025). Acerca de Azure Update Manager. https://learn.microsoft.com/es-es/azure/update-manager/overview

NASA. (s. f.). Technology readiness levels. https://www.nasa.gov/directorates/somd/space-communications-navigation-program/technology-readiness-levels/

NinjaOne. (2024, 15 de enero). NinjaOne, elegido por los principales sitios de reseñas como la mejor plataforma de gestión de endpoints. https://www.ninjaone.com/es/press/ninjaone-

elegido-mejor-plataforma-gestion-de-endpoints-por-principales-sitios-de-resenas-de-clientes/

NinjaOne. (s. f.). NinjaOne - top-rated UEM & IT management software. https://www.ninjaone.com/

Oracle. (2025). Alta disponibilidad en la nube. https://docs.oracle.com/es-ww/iaas/Content/cloud-adoption-framework/high-availability.htm

Pensemos S.A. (2024). Suite visión empresarial. https://pensemos.com/suite-vision-empresarial/

Platform Engineering. (2025). From pipelines to platforms: The evolution of CI/CD in enterprise DevOps. https://platformengineering.com/ci-cd-pipelines/from-pipelines-to-platforms-the-evolution-of-ci-cd-in-enterprise-devops/

Red Hat. (2022). La integración y la distribución continuas (CI/CD). https://www.redhat.com/es/topics/devops/what-is-ci-cd

Red Hat. (s. f.). Red Hat Ansible automation platform. https://www.redhat.com/es/technologies/management/ansible

ServiceNow. (2025). ¿Qué es CI/CD? https://www.servicenow.com/latam/products/devops/what-is-cicd.html

TheirStack. (s. f.). Empresas que usan Jenkins. https://theirstack.com/es/technology/jenkins

Universidad Republicana. (2024). Integración y despliegue continuo con DevOps como paradigma. https://ojs.urepublicana.edu.co/index.php/ingenieria/article/download/881/636