

Universidad Internacional de La Rioja

Escuela Superior de Ingeniería y Tecnología

Máster Universitario en Desarrollo y Operaciones (DevOps)

Implementar el proceso DevSecOps automatizado con Packer y Terraform para multi plataformas

|  |  |
| --- | --- |
| Trabajo fin de estudio presentado por: | Armijo Quito Jorge Luis |
| Tipo de trabajo: | Desarrollo Práctico |
| Director/a: | María Gema Cano López |
| Fecha: | 16/07/2025 |

Resumen

El presente Trabajo Fin de Máster tiene como objetivo definir e implementar un proceso DevSecOps automatizado que permita gestionar la infraestructura tecnológica de forma segura, portable y escalable sobre múltiples plataformas sean estas locales o públicas como AWS, Azure. En un contexto donde las instituciones enfrentan una creciente presión por mejorar la seguridad sin comprometer la agilidad, surge la necesidad de integrar controles desde las etapas iniciales del ciclo de vida de la infraestructura, alineando la automatización con políticas de cumplimiento y mejores prácticas.

El problema identificado radica en la carencia de modelos automatizados que combinen la generación segura de entornos con validaciones continuas de configuración y seguridad, especialmente en escenarios multi nube. Para abordar esta necesidad, se propone un flujo de trabajo que utiliza Packer para la construcción de imágenes base endurecidas, Terraform para la provisión y gestión de la infraestructura como código (IaC), y Ansible para la aplicación automatizada de políticas de endurecimiento de seguridad. Estos procesos son orquestados mediante pipelines CI/CD, los cuales incorporan herramientas de escaneo de vulnerabilidades y validación de seguridad, garantizando la consistencia y el cumplimiento normativo de las configuraciones desplegadas.

La metodología aplicada para el desarrollo de este TFM está basada en la integración de varias herramientas open source en un entorno de pruebas controlado, lo que permite replicar escenarios realistas para validar la eficiencia y seguridad del proceso. Como resultado, se espera lograr una solución técnica replicable que mejore los tiempos de provisión, reduzca errores humanos y fortalezca la seguridad.

Este trabajo constituye una contribución práctica al marco de adopción de DevSecOps, ofreciendo una guía práctica estructurada y automatizada aplicable tanto en infraestructuras locales (on-premise) como en entornos de nube, promoviendo así la sostenibilidad operativa, la eficiencia técnica y la mejora continua en la gestión de entornos tecnológicos.

**Palabras clave:** (DevSecOps, Infraestructura como Código, Automatización, MultiNube, Hardening)

Abstract

This Master's Thesis aims to define and implement an automated DevSecOps process to manage IT infrastructure in a secure, portable, and scalable way across multiple platforms, including AWS, Azure, and VMware. In a context where organizations face increasing pressure to improve security without compromising agility, it becomes essential to integrate controls early in the infrastructure lifecycle, aligning automation with compliance policies and best practices.

The core problem addressed in this work is the lack of automated models that combine secure environment provisioning with continuous security and configuration validations, especially in multi nube scenarios. To address this, a workflow is proposed using Packer to build hardened base images, Terraform to provision and manage infrastructure as code (IaC), and Ansible to apply automated hardening policies. These processes are orchestrated through CI/CD pipelines that integrate vulnerability scanning and configuration validation tools, ensuring consistency and compliance across deployed environments.

The methodology is based on the integration of open-source tools within a controlled testing environment, enabling the simulation of realistic deployment scenarios to assess the efficiency and security of the process. As an outcome, the project aims to deliver a replicable technical solution that enhances provisioning times, reduces human errors, and strengthens the organization’s overall security posture.

This work contributes to the practical adoption of DevSecOps by offering a structured, automated, and portable implementation framework. It is applicable in both on-premise and cloud environments and promotes operational sustainability, technical efficiency, and continuous improvement in the management of technological infrastructures.

**Keywords**: (DevSecOps, Infraestructua as Code, Automation, Multi nube, Hardening)

Índice de contenidos

[1. Introducción 9](#_Toc203510969)

[1.1. Justificación del trabajo 9](#_Toc203510970)

[1.2. Planteamiento del problema 10](#_Toc203510971)

[1.3. Estructura de la memoria 12](#_Toc203510972)

[2. Contexto y estado del arte 16](#_Toc203510973)

[2.1. Contextualización y antecedentes 16](#_Toc203510974)

[2.1.1. Evolución de DevOps hacia DevSecOps 18](#_Toc203510975)

[2.1.2. Infraestructura como Código (IaC): Automatización y gestión de configuraciones 20](#_Toc203510976)

[2.1.3. Seguridad en entornos multi nube: Desafíos y estrategias 22](#_Toc203510977)

[2.1.4. Herramientas y tecnologías relevantes 23](#_Toc203510978)

[2.2. Trabajos relacionados 24](#_Toc203510979)

[2.2.1. Aportes académicos 27](#_Toc203510980)

[2.2.2. Soluciones comerciales e industriales 34](#_Toc203510981)

[2.2.3. Análisis crítico y aportación del presente trabajo 36](#_Toc203510982)

[2.3. Conclusiones del estado del arte 37](#_Toc203510983)

[3. Objetivos y metodología de trabajo 38](#_Toc203510984)

[3.1. Objetivo general 38](#_Toc203510985)

[3.2. Objetivos específicos 38](#_Toc203510986)

[3.3. Metodología del trabajo 40](#_Toc203510987)

[3.3.1. Fase I – Diseño del flujo DevSecOps 41](#_Toc203510988)

[3.3.2. Fase II – Creación de imágenes base con Packer 43](#_Toc203510989)

[3.3.3. Fase III – Gestión de infraestructura con Terraform 44](#_Toc203510990)

[3.3.4. Fase IV – Aplicación de hardening con Ansible 46](#_Toc203510991)

[3.3.5. Fase V – Orquestación en pipeline CI/CD 48](#_Toc203510992)

[3.3.6. Fase VI – Despliegue continuo y validación técnica. 52](#_Toc203510993)

[3.3.7. Fase VII – Conclusión técnica del desarrollo 54](#_Toc203510994)

[4. Desarrollo específico de la contribución 55](#_Toc203510995)

[4.1. Planificación / Análisis / Requisitos 56](#_Toc203510996)

[4.1.1. Identificación del problema y contexto de uso 56](#_Toc203510997)

[4.1.2. Tecnologías seleccionadas y justificación 57](#_Toc203510998)

[4.1.3. Organización del desarrollo 57](#_Toc203510999)

[4.1.4. Técnicas utilizadas 58](#_Toc203511000)

[4.1.5. Instrumentos de seguimiento y evaluación 59](#_Toc203511001)

[4.2. Descripción del sistema desarrollado / Implementación 59](#_Toc203511002)

[4.2.1. Arquitectura multicapa y contenedores 60](#_Toc203511003)

[4.2.2. Construcción de imagen base con Packer 61](#_Toc203511004)

[4.2.3. Provisionamiento con Terraform 62](#_Toc203511005)

[4.2.4. Construcción de imagen Docker para compilación Java 62](#_Toc203511006)

[4.2.5. Pipeline CI/CD modular en Azure DevOps 63](#_Toc203511007)

[4.2.6. Despliegue continúo automatizado 63](#_Toc203511008)

[4.2.7. Compatibilidad multicloud en AWS y Azure 64](#_Toc203511009)

[4.3. Evaluación 65](#_Toc203511010)

[4.3.1. Usabilidad y aplicabilidad de la solución 66](#_Toc203511011)

[4.3.2. Pruebas unitarias, de integración y validación funcional 66](#_Toc203511012)

[4.3.3. Análisis de cumplimiento de objetivos 67](#_Toc203511013)

[4.3.4. Valor añadido de la solución desarrollada 68](#_Toc203511014)

[4.4. Limitaciones del sistema desarrollado 69](#_Toc203511015)

[4.4.1. Alcance limitado del entorno de despliegue 70](#_Toc203511016)

[4.4.2. Curva de aprendizaje y dependencia técnica 70](#_Toc203511017)

[4.4.3. Ausencia de herramientas de monitoreo centralizado 71](#_Toc203511018)

[4.4.4. Seguridad en etapas avanzadas 71](#_Toc203511019)

[4.4.5. Infraestructura sin alta disponibilidad (HA) 71](#_Toc203511020)

[4.4.6. Dependencia de servicios cloud externos 71](#_Toc203511021)

[5. Conclusiones y trabajo futuro 72](#_Toc203511022)

[5.1. Conclusiones 72](#_Toc203511023)

[5.2. Líneas de trabajo futuro 73](#_Toc203511024)

[Referencias bibliográficas 74](#_Toc203511025)

[Anexo A. Repositorio de Código 75](#_Toc203511026)

[Anexo B. Estructura del proyecto 76](#_Toc203511027)

[Anexo C. Capturas de imagen del proyecto 76](#_Toc203511028)

Índice de figuras

[Ilustración 1 - Flujo del pipeline CI/CD Fuente: Elaboración propia 51](#_Toc202536299)

[Ilustración 2 - Diagrama de arquitectura lógica del sistema Fuente: Elaboración propia 60](#_Toc202536300)

[Ilustración 3 - Arquitectura del proyecto TFM Fuente: Elaboración propia 64](#_Toc202536301)

[Ilustración 4 - Tiempos promedio por fase del pipeline Fuente: Elaboración propia 68](#_Toc202536302)

[Ilustración 5 - Estructura archivos Proyecto Fuente: Elaboración propia 75](#_Toc202536303)

Índice de tablas

[Tabla 1 - Evolución de DevOps hacia DevSecOps Fuente: Elaboración propia 19](#_Toc202536304)

[Tabla 2 - Infraestructura como Código: Automatización y gestión de configuraciones Fuente: Elaboración propia 21](#_Toc202536305)

[Tabla 3 - Fases del pipeline CI/CD con herramientas aplicadas Fuente: Elaboración propia 51](#_Toc202536306)

[Tabla 4 - Validación del sistema Fuente: Elaboración propia 53](#_Toc202536307)

[Tabla 5 - Cronograma del desarrollo Fuente: Elaboración propia 58](#_Toc202536308)

[Tabla 6 - Comparación entre despliegue manual vs automatizado Fuente: Elaboración propia 65](#_Toc202536309)

[Tabla 7 - Cumplimiento de objetivos específicos vs resultados Fuente: Elaboración propia 67](#_Toc202536310)

# Introducción

En la actualidad, las instituciones públicas o privadas enfrentan el desafío de gestionar infraestructuras tecnológicas complejas que requieren ser altamente seguras, escalables y adaptables a múltiples plataformas. La adopción de prácticas como DevOps y la implementación de Infraestructura como Código (IaC) han emergido como soluciones efectivas para automatizar y estandarizar la provisión y configuración de recursos tecnológicos. Sin embargo, la integración de la seguridad desde las fases iniciales del desarrollo y despliegue de infraestructura, conocida como DevSecOps, se ha convertido en una necesidad imperante para garantizar la protección de los sistemas en entornos multi nube.

El presente trabajo se centra en la definición e implementación de un proceso DevSecOps automatizado que integre herramientas como Packer y Terraform para facilitar la creación y configuración segura de infraestructura en múltiples plataformas, incluyendo nubes públicas como AWS, Azure y nubes privadas como VMware. La propuesta busca garantizar que las soluciones implementadas sean portables, escalables y cumplan con los estándares de seguridad, permitiendo una gestión más eficiente y confiable de los entornos tecnológicos.

## Justificación del trabajo

Este trabajo aborda la falta de flujos automatizados que integren de forma temprana y coherente las prácticas de seguridad en el ciclo de vida de la infraestructura tecnológica, especialmente en entornos multi nube. Muchas instituciones públicas y privadas, al adoptar modelos DevOps, siempre se enfocan en priorizar la velocidad de entrega y automatización sin considerar desde el inicio medidas de seguridad que aseguren la integridad, disponibilidad y confidencialidad de los sistemas. Esto ha derivado en configuraciones inseguras, vulnerabilidades no controladas y deficiencias en los sistemas informáticos que pueden dejar deuda técnica en los ambientes de producción sin una solución a corto o mediano plazo.

Entre una de las principales causas de este problema es la relación con la falta de madurez en los procesos de automatización adoptados y sin un claro enfoque en seguridad (DevSecOps) a esta problemática también se une la escasa o ninguna integración entre los equipos de desarrollo, operaciones y seguridad. De la experiencia adquirida en el transcurso de los años se puede decir que a existencia de herramientas como Terraform, Packer y Ansible facilitan el abordar técnicamente estos retos sociales y técnicos, pero para una adopción efectiva se requiere de la definición de flujos claros, repetibles y auditables que estén asociados con los procesos internos de las instituciones sean estas públicas o privadas. En este contexto se podría mencionar estudios como los de Sharma et al. (2020) y Williams & Fernández (2019) en donde demuestran que la automatización con enfoque DevSecOps puede reducir significativamente las vulnerabilidades operativas, mejorando la eficiencia y reduciendo el riesgo.

La importancia del desarrollo de este trabajo de fin de máster está orientado en base a las necesidades identificadas y sobre la experiencia obtenida en el transcurso de los años como consultor experto en sistemas Linux y servidores de aplicaciones, así como empleado de una de las instituciones más relevantes en el sector público para la recaudación de impuestos en el Ecuador, así también, para la comunidad educativa, científica y empresarial y radica en que se ofrece una alternativa técnica totalmente replicable para abordar un problema vigente en la actualidad en que todos buscan ejecutar una transformación digital de instituciones ya sean públicas o privadas. Considerando una perspectiva académica, el presente trabajo aportará un marco metodológico para enseñar el cómo aplicar los principios de seguridad desde el diseño (“security by design”) en el despliegue de infraestructuras automatizadas, contribuyendo así a la consolidación de prácticas DevSecOps.

## Planteamiento del problema

En la actualidad, las instituciones públicas y privadas enfrentan el desafío constante de la transformación digital y poder adaptar sus procesos de puesta en producción en una sinergia constante con su infraestructura tecnológica sea esta privada, pública o en muchos casos como se puede ver ahora en ecosistemas híbridos, donde la seguridad, la velocidad de despliegue y la escalabilidad no son opcionales, sino que son requisitos fundamentales. En este contexto, los enfoques tradicionales de gestión del ciclo de la infraestructura han comenzado a mostrar sus limitaciones en donde los despliegues manuales son más propensos a errores humanos, los tiempos de aprovisionamiento son extensos y desgastantes, así como, los entornos tienen más susceptibles a incurrir en inconsistencias entre entornos de desarrollo, pruebas, certificación, y producción, en donde es evidente que no existe un control sobre las configuraciones recayendo en que los ambientes mencionados no cumplen con estándares mínimos de configuración y menos aún de seguridad.

Aunque la adopción de metodologías DevOps ha permitido avanzar hacia procesos más ágiles y automatizados, en muchos casos estos avances no han ido acompañados de una integración efectiva de la seguridad. Con base a la experiencia adquirida en el transcurso de los años es evidente que esta ausencia de controles desde etapas tempranas del desarrollo de aplicaciones genera vulnerabilidades que tienen un alto porcentaje de ser explotadas, afectando de forma exponencial a la disponibilidad, confidencialidad e integridad de los sistemas. En escenarios multi nube que cada vez más comunes por sus ventajas en resiliencia y flexibilidad, así como también, la situación se agrava debido a la heterogeneidad de servicios, configuraciones y políticas de cada proveedor, de ahí la importancia del desarrollo de este trabajo en el cual se trata de cubrir esta brecha de conocimientos y poder ser un aporte efectivo y eficaz para los desarrolladores, ingenieros de pruebas y seguridad y administradores de infraestructura que buscan ese salto en la transformación digital de sus instituciones.

El problema identificado y el cual se quiere solventar con este trabajo fin de máster es la falta de conocimiento en identificar y definir un flujo automatizado, seguro y replicable el que permita crear, configurar y validar infraestructura tecnológica desde un enfoque DevSecOps, asegurando tanto la velocidad de despliegue como la conformidad con normativas de seguridad. Con base a la experiencia adquirida a lo largo de los años se evidencia que es una necesidad real en instituciones públicas y privadas de contar con procesos que no solo automaticen el aprovisionamiento, sino que integren controles de seguridad de forma nativa, desde la construcción de imágenes base hasta su despliegue en la nube o en entornos on-premise.

El presente trabajo de fin de máster se lo plantea como solución la implementación de un proceso DevSecOps automatizado, apoyado en herramientas de Infraestructura como Código (IaC) como Packer, Terraform y Ansible, con orquestación a través de pipelines CI/CD. La finalidad es demostrar que es posible construir un flujo técnico que permita generar imágenes base endurecidas, y utilizables en los ambientes de desarrollo, pruebas, producción y que sean completamente seguras para sean aplicadas en entornos múltiples.

## Estructura de la memoria

Este Trabajo Fin de Máster ha sido estructurado cuidadosamente para responder tanto a los retos técnicos actuales como a las necesidades operativas que enfrentan las organizaciones en el despliegue seguro y eficiente de infraestructuras tecnológicas. La evolución constante de los entornos informáticos, especialmente en escenarios híbridos y multicloud, exige que los profesionales dominen metodologías de automatización, aprovisionamiento reproducible y seguridad embebida desde el diseño. En este contexto, la presente propuesta se construye como una guía integral y aplicada que puede ser utilizada en diferentes fases del ciclo de vida de la infraestructura: desarrollo, pruebas, certificación y producción.

La solución aquí planteada se alinea plenamente con los objetivos del Máster Universitario en Desarrollo y Operaciones (DevOps) de la UNIR, al integrar herramientas ampliamente adoptadas en la industria como Packer, Terraform, Ansible y Azure DevOps. Su implementación práctica responde a principios clave como la infraestructura como código, la automatización del cumplimiento normativo y la validación continua de entornos. Estos principios se aplican de forma orquestada mediante pipelines CI/CD, permitiendo no solo acelerar el proceso de despliegue, sino también mejorar la trazabilidad, reducir errores humanos y garantizar la consistencia de las configuraciones.

Este trabajo no se limita a presentar un ejercicio técnico, sino que pretende convertirse en un recurso pedagógico y operativo útil para profesionales, docentes y equipos técnicos que enfrentan la transformación digital de sus organizaciones. Al basarse en herramientas open source y flujos modulares, la solución es portable, replicable y adaptable a distintos niveles de madurez tecnológica. Su valor reside en ofrecer una alternativa concreta frente a la brecha entre teoría y práctica que aún persiste en muchos entornos institucionales, especialmente del sector público.

Cada capítulo del documento ha sido diseñado para construir una narrativa coherente: desde la contextualización del problema y el análisis del estado del arte, pasando por la definición metodológica y el desarrollo técnico, hasta la evaluación, reflexión y líneas futuras de mejora. Este enfoque progresivo permitirá al lector del presente documento el poder comprender en profundidad no solo lo que se ha desarrollado, sino por qué se lo ha realizado y el cómo este puede adaptarse a su propio entorno.

**Introducción. -** Esta sección introduce el contexto general del Trabajo Fin de Máster, partiendo de los desafíos actuales en la gestión de infraestructuras tecnológicas, especialmente en entornos híbridos y multi nube. La necesidad de automatizar procesos, asegurar configuraciones desde el diseño y reducir errores humanos ha evidenciado las limitaciones de los enfoques tradicionales.

La propuesta nace de experiencias profesionales en instituciones donde aún persisten prácticas manuales y poco seguras. En respuesta, se plantea una solución práctica que integra herramientas open source y principios DevSecOps para lograr entornos seguros, portables y reproducibles.

Este capítulo presenta también la motivación y relevancia del trabajo en contextos académicos y profesionales, destacando su aplicabilidad en equipos con distintos niveles de madurez tecnológica. Finalmente, se definen los objetivos generales que guían el desarrollo del proyecto y se anticipa la estructura del documento, proporcionando al lector una visión clara de los contenidos y su progresión lógica.

**Contexto y estado del arte. -** En esta sección se profundiza en los fundamentos teóricos y técnicos que dan soporte al desarrollo del trabajo, estableciendo un marco de referencia riguroso y actualizado. Se abordan conceptos clave como DevOps, DevSecOps, Infraestructura como Código (IaC), automatización de procesos y seguridad embebida en entornos tecnológicos complejos, con especial énfasis en arquitecturas híbridas y multi nube. Estos elementos son analizados desde una perspectiva crítica y aplicada, considerando su evolución, su adopción en la industria y su relevancia en el proceso de transformación digital.

Asimismo, se realiza una revisión de investigaciones previas, estudios académicos y experiencias industriales que han enfrentado problemáticas similares, con el objetivo de identificar buenas prácticas, metodologías consolidadas y áreas de oportunidad. Esta exploración permite no solo contextualizar el estado actual del conocimiento, sino también detectar vacíos técnicos y operativos que justifican la necesidad de la solución propuesta en este Trabajo Fin de Máster, reforzando su valor como contribución original y replicable.

**Objetivos y metodología. -** En esta sección se define con precisión el objetivo general del trabajo de fin de máster y los objetivos específicos que orientan su desarrollo. Posteriormente, se describe la metodología empleada, explicando las herramientas seleccionadas (Packer, Terraform, Ansible, flujos CI/CD), el enfoque de automatización adoptado, y la estrategia de validación aplicada. Se detalla cómo se estructuró el proceso de trabajo para garantizar la portabilidad, la seguridad y la reproducibilidad de los resultados.

**Desarrollo de la contribución. –** En esta sección se establecen con claridad tanto el objetivo general del Trabajo Fin de Máster como los objetivos específicos que guían su desarrollo técnico y metodológico. El enfoque adoptado parte de una necesidad real: implementar un flujo DevSecOps automatizado, seguro y replicable, aplicable a entornos híbridos y multi nube. Para alcanzar dicho propósito, se definen metas concretas orientadas a integrar herramientas de Infraestructura como Código (IaC), aplicar políticas de seguridad desde el diseño, y validar la efectividad operativa mediante pruebas reproducibles en escenarios reales.

A continuación, se describe detalladamente la metodología empleada. Esta se basa en un enfoque experimental, secuencial e iterativo, que permite combinar el uso de tecnologías open source como Packer para la creación de imágenes base, Terraform para el aprovisionamiento automatizado, y Ansible para el hardening de sistemas con flujos de integración y entrega continua (CI/CD) implementados en Azure DevOps.

La estrategia metodológica no solo busca automatizar tareas, sino garantizar que los resultados sean portables, seguros, auditables y fácilmente reproducibles en diferentes plataformas. Cada fase del proceso ha sido estructurada con criterios de trazabilidad, separación de responsabilidades y validación progresiva. Este enfoque permite evaluar la calidad técnica del sistema propuesto, al tiempo que habilita su adaptación en instituciones con diversos niveles de madurez tecnológica, consolidando así su valor como solución práctica.

**Conclusiones y líneas de trabajo futuro. -** En el capítulo final se presenta un análisis reflexivo sobre el trabajo desarrollado, poniendo en perspectiva los resultados obtenidos y evaluando el grado de cumplimiento de los objetivos inicialmente planteados. Esta sección no solo sintetiza los principales logros alcanzados durante el desarrollo del sistema DevSecOps automatizado, sino que también permite valorar su aplicabilidad en escenarios reales y su alineación con los principios de portabilidad, seguridad y eficiencia operativa promovidos a lo largo del proyecto.

Además del balance técnico, se identifican posibles limitaciones asociadas al entorno de pruebas utilizado, a la cobertura de validaciones de seguridad y al alcance de las herramientas aplicadas. Reconocer estos límites resulta fundamental para proponer líneas de mejora que permitan evolucionar la solución hacia contextos de mayor complejidad o exigencia. Entre las posibles ampliaciones se considera la integración de mecanismos formales de cumplimiento normativo (por ejemplo, ISO/IEC 27001 o NIST SP 800-53), el despliegue sobre plataformas de orquestación de contenedores como Kubernetes, y la extensión del flujo a organizaciones con estructuras operativas más robustas o distribuidas.

Este último capítulo busca, por tanto, consolidar los aportes del trabajo no solo desde un punto de vista técnico, sino también estratégico, ofreciendo una visión a futuro sobre cómo continuar perfeccionando la solución propuesta y adaptándola a nuevos desafíos del ecosistema tecnológico.

# Contexto y estado del arte

En esta sección se establece el marco conceptual y práctico esencial para entender el enfoque DevSecOps propuesto. Se revisan críticamente conceptos clave como DevOps, su evolución hacia DevSecOps, la Infraestructura como Código (IaC), y las estrategias de seguridad en ambientes multi nube. Además, se identifican las herramientas más importantes que apoyan esta metodología.

En la subsección 2.1, "Contextualización y antecedentes", se describe cómo las organizaciones han avanzado desde modelos tradicionales hacia enfoques ágiles y seguros como DevSecOps, destacando la integración temprana de la seguridad y los desafíos específicos en la gestión de infraestructuras híbridas y multi nube.

La subsección 2.2, "Trabajos relacionados", presenta una revisión breve de aportes académicos y soluciones comerciales destacadas, identificando buenas prácticas y limitaciones que respaldan la relevancia de la propuesta desarrollada en este trabajo.

Finalmente, la subsección 2.3, "Conclusiones del estado del arte", se resumen los hallazgos clave, subrayando la importancia de automatizar la gestión segura y escalable de infraestructuras tecnológicas, especialmente para instituciones con limitaciones técnicas.

## Contextualización y antecedentes

La evolución de la gestión de infraestructura tecnológica ha sido uno de los ejes transformadores más relevantes de la última década en el ámbito de las tecnologías de la información. La necesidad de responder con agilidad a los cambios del mercado, asegurar la continuidad operativa y garantizar la protección de activos digitales ha llevado a las organizaciones a repensar sus modelos de aprovisionamiento, despliegue y operación de sistemas. En este contexto, han surgido paradigmas como DevOps y, más recientemente, DevSecOps, que no solo proponen una forma diferente de trabajar, sino también una transformación cultural y técnica profunda en las organizaciones.

Las arquitecturas modernas, especialmente aquellas basadas en servicios distribuidos, entornos híbridos o infraestructura multicloud, han desafiado los enfoques tradicionales de gestión y operación. En muchos casos, el despliegue de aplicaciones y servidores sigue dependiendo de procesos manuales, fragmentados y difíciles de auditar. Esta situación no solo genera cuellos de botella operativos, sino que incrementa significativamente los riesgos de errores humanos, brechas de configuración y vulnerabilidades de seguridad. Para quienes hemos trabajado en entornos reales y particularmente en instituciones públicas o grandes estructuras con procesos rígidos estas deficiencias no son una abstracción teórica, sino una realidad operativa con consecuencias concretas.

En respuesta a este escenario, el enfoque DevSecOps emerge como una evolución natural del modelo DevOps, integrando la seguridad como una responsabilidad compartida desde las primeras fases del ciclo de vida del software e infraestructura. DevSecOps no consiste en “agregar” controles de seguridad como una capa posterior al desarrollo, sino en “incrustarlos” en cada fase del proceso: desde la planificación, la construcción de artefactos, el aprovisionamiento de entornos, hasta las pruebas, validaciones y la puesta en producción. Este cambio de mentalidad es lo que conocemos como “shift left security”, una práctica que, según Williams y Fernández (2019), permite reducir el coste de las vulnerabilidades y mejorar la resiliencia operativa de los sistemas.

Paralelamente, el concepto de Infraestructura como Código (IaC) ha revolucionado la forma en que se define, despliega y mantiene la infraestructura tecnológica. IaC permite describir los recursos de una organización (máquinas virtuales, redes, bases de datos, reglas de seguridad, etc.) mediante archivos legibles por humanos y gestionables en sistemas de control de versiones. Herramientas como Terraform, Packer, Ansible, Puppet o Chef se han consolidado como estándares de facto para implementar IaC en entornos reales. Estas tecnologías permiten no solo automatizar tareas repetitivas, sino también lograr configuraciones reproducibles, auditables y fácilmente escalables.

Sin embargo, IaC por sí sola no resuelve el problema de la seguridad. La automatización sin validación puede generar errores a gran escala. Por esta razón, surge la necesidad de combinar IaC con prácticas DevSecOps, donde cada plantilla, cada recurso desplegado, cada configuración aplicada, sea sometida a análisis de cumplimiento, pruebas automatizadas y escaneo de vulnerabilidades. Esta integración de IaC y DevSecOps constituye, hoy en día, uno de los pilares fundamentales para alcanzar una gestión moderna, segura y eficiente de entornos tecnológicos.

En los entornos multicloud, este desafío se agudiza. La heterogeneidad de servicios, APIs, modelos de autenticación y mecanismos de gobernanza entre proveedores como AWS, Azure o Google Cloud genera una capa adicional de complejidad. En muchas organizaciones particularmente aquellas que aún están transitando hacia una madurez operativa en la nube esto se traduce en configuraciones inconsistentes, políticas dispares, y una dificultad real para mantener una postura de seguridad homogénea. Como señala Paul (2024), la gestión segura de infraestructuras multicloud requiere no solo automatización, sino estandarización, visibilidad y capacidad de respuesta continua ante cambios en el entorno.

Todo este contexto justifica la necesidad de una propuesta técnica práctica que no se limite a teorizar sobre estas herramientas y metodologías, sino que las integre en un flujo automatizado, probado y replicable. Este Trabajo Fin de Máster parte precisamente de esa necesidad observada en el campo donde el demostrar que es posible implementar un proceso DevSecOps real, utilizando tecnologías open source, en entornos controlados pero representativos de organizaciones modernas. La solución aquí propuesta no pretende ser universal ni definitiva, pero sí suficientemente robusta, adaptable y accesible como para servir de base en proyectos institucionales y académicos que busquen avanzar hacia una gestión segura, ágil y automatizada de su infraestructura tecnológica.

### Evolución de DevOps hacia DevSecOps

La evolución de las metodologías de desarrollo de software en la última década ha estado marcada por la búsqueda constante de eficiencia, agilidad y reducción de errores. Durante muchos años, los equipos de desarrollo y operaciones trabajaban de forma aislada, lo que generaba fricciones significativas en los ciclos de entrega. Este modelo tradicional, conocido como "desarrollo en silos", dificultaba la colaboración, aumentaba los tiempos de despliegue y generaba inconsistencias entre los entornos de prueba y producción. Como respuesta a esta fragmentación surgió DevOps, una metodología que busca unir a ambos mundos mediante prácticas colaborativas, automatización de procesos y mejora continua.

DevOps propuso una nueva forma de concebir el ciclo de vida del software: desde el código fuente hasta el despliegue en producción, todo debía ser fluido, integrado y automatizado. Esta visión se materializó en la implementación de pipelines de integración y entrega continua (CI/CD), infraestructura como código, pruebas automatizadas y monitoreo proactivo, predictivo asociado también como Observabilidad de la Infraestructura en TI. Sin embargo, con el paso del tiempo, la propia eficiencia de DevOps comenzó a revelar una debilidad estructural: la seguridad no estaba plenamente integrada, y a menudo se relegaba a fases finales del ciclo o a auditorías posteriores al despliegue.

En un contexto donde las amenazas cibernéticas aumentan en volumen y sofisticación, esta omisión se convirtió en un riesgo crítico. Así emergió el enfoque DevSecOps, que no es una metodología distinta, sino una evolución natural de DevOps. DevSecOps extiende la cultura de colaboración para incluir también a los equipos de seguridad, haciendo que la protección de los sistemas sea una responsabilidad compartida desde el primer momento.

Este modelo promueve la integración temprana de herramientas de análisis estático (SAST) y dinámico (DAST), pruebas automatizadas de vulnerabilidades, escaneos de dependencias, revisión de políticas de cumplimiento y análisis de infraestructura como código. Como señala Sandu (2021), DevSecOps permite integrar la seguridad sin sacrificar la velocidad del desarrollo, y Adewusi (2023) destaca su impacto positivo en la resiliencia de las aplicaciones al detectar y corregir fallos desde fases iniciales.

Además, DevSecOps trasciende el software. En proyectos como el que se aborda en este Trabajo Fin de Máster, la seguridad también se incorpora en la infraestructura subyacente, mediante la automatización de configuraciones seguras y la implementación de flujos que respetan los principios de "security by design". Esta convergencia entre desarrollo, operaciones y seguridad representa un cambio de paradigma que ya está siendo adoptado por empresas e instituciones de todo tipo, y se alinea plenamente con los objetivos estratégicos del presente trabajo.

La siguiente tabla tiene como propósito resaltar las diferencias fundamentales entre los enfoques DevOps y DevSecOps, considerando aspectos como la automatización, la integración de seguridad, la responsabilidad organizacional y el conjunto de herramientas asociadas. Esta comparativa permite contextualizar la evolución hacia un modelo donde la seguridad no es una fase final, sino un componente transversal que guía el diseño y la operación desde el inicio del ciclo de vida del software.

|  |  |  |
| --- | --- | --- |
| **Dimensión** | **DevOps** | **DevSecOps** |
| Enfoque general | Colaboración entre desarrollo y operaciones para acelerar entregas | Integración de seguridad desde el diseño en todo el ciclo de vida |
| Objetivo principal | Entrega continua de software funcional y estable | Entrega continua de software seguro y confiable |
| Seguridad | Implementada en fases finales o posterior al despliegue | Integrada desde fases iniciales de desarrollo |
| Automatización | Alta, centrada en integración y despliegue continuo | Alta, incluyendo escaneos, validación de IaC, cumplimiento |
| Responsabilidad | Desarrolladores y operadores | Desarrolladores, operadores y especialistas en seguridad |
| Herramientas comunes | Jenkins, Git, Docker, Kubernetes, Ansible | Jenkins, Git, Docker, Kubernetes, Ansible, Trivy, tfsec |
| Integración de seguridad | Generalmente posterior al desarrollo | Incorporada desde el inicio (shift-left security) |
| Cultura organizacional | Colaborativa entre Dev y Ops | Cultura colaborativa ampliada con prácticas de seguridad |

Tabla 1 - Evolución de DevOps hacia DevSecOps Fuente: Elaboración propia

### Infraestructura como Código (IaC): Automatización y gestión de configuraciones

La Infraestructura como Código (IaC) es una de las prácticas fundamentales que ha transformado la gestión de sistemas en la última década. Consiste en definir y aprovisionar recursos de infraestructura como servidores, redes, bases de datos, reglas de seguridad o balanceadores de carga utilizando archivos de configuración legibles por máquina, versionables y ejecutables. Esta estrategia permite gestionar entornos tecnológicos de forma similar al desarrollo de software: con control de versiones, pruebas, revisiones de cambios y despliegues automatizados.

En lugar de configurar manualmente cada recurso, los equipos describen su infraestructura deseada mediante lenguajes declarativos o scripts, que pueden ser auditados, reutilizados y aplicados en diferentes entornos con un alto grado de consistencia. Esta aproximación facilita la reproducibilidad entre ambientes, reduce el tiempo necesario para levantar nuevos entornos y elimina muchos de los errores humanos asociados al trabajo manual.

Herramientas como Terraform, Ansible, Puppet y Chef se han consolidado como estándares en la implementación de IaC. Terraform, por ejemplo, permite describir infraestructura de múltiples proveedores (AWS, Azure, Google Cloud, VMware, etc.) a través de archivos .tf, gestionando el ciclo de vida completo de los recursos desde un estado deseado. Ansible, por su parte, se orienta a la automatización de configuraciones y tareas post-aprovisionamiento, como el despliegue de software, la aplicación de políticas de hardening o la gestión de usuarios. Puppet y Chef, aunque con un enfoque más complejo y robusto para grandes organizaciones, también permiten administrar configuraciones de forma idempotente y controlada.

Como señala Özdoğan et al. (2023), la adopción de IaC mejora significativamente la eficiencia operativa, permite escalar sistemas de forma más ordenada y fortalece la gobernanza técnica al ofrecer trazabilidad sobre cada cambio realizado. Sin embargo, su uso también introduce nuevos desafíos, especialmente en el ámbito de la seguridad y la validación semántica.

Los archivos de configuración, al no estar versionados, automatizados y ejecutarse sin supervisión directa, pueden contener errores lógicos, referencias inseguras, credenciales mal gestionadas o configuraciones que abren puertas innecesarias en los sistemas. Por ello, es imprescindible aplicar análisis estático y dinámico sobre los scripts IaC, utilizando herramientas que permitan identificar riesgos antes de que los recursos sean aprovisionados.

En el marco del presente trabajo, la IaC no solo se utiliza para facilitar la automatización, sino como una pieza clave del enfoque DevSecOps, donde cada componente de infraestructura es tratado como código auditado, versionado y validado, garantizando así que las plataformas desplegadas no solo sean funcionales, sino también seguras y replicables desde el diseño.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Herramienta** | **Tipo** | **Uso principal** | **Ventajas** | **Limitaciones** |
| Terraform | Infraestructura como Código | Provisionamiento de recursos en nube y on-premise | Declarativo, multicloud, reutilizable, soporta módulos | Curva de aprendizaje en sintaxis HCL y manejo de estado |
| Ansible | Automatización de configuración | Configuración, instalación de software, hardening de sistemas | Fácil de aprender, sin agentes, muy flexible | Menos robusto para grandes infraestructuras |
| Puppet | Gestión de configuración | Automatización de configuración basada en manifiestos | Madurez y comunidad activa, control granular | Requiere agentes, complejidad creciente |
| Chef | Gestión de configuración | Automatización de infraestructura orientada a recetas | Alta integración con sistemas legacy | Dependencia de su ecosistema, curva de aprendizaje |
| Packer | Creación de imágenes | Generación de imágenes consistentes para múltiples entornos | Portabilidad, integración con CI/CD, personalización | No realiza despliegue, solo construcción de imágenes |

Tabla 2 - Infraestructura como Código: Automatización y gestión de configuraciones Fuente: Elaboración propia

### Seguridad en entornos multi nube: Desafíos y estrategias

La adopción de estrategias multi nube se ha convertido en una práctica cada vez más común entre organizaciones que buscan optimizar recursos, garantizar la continuidad del negocio y evitar la dependencia de un solo proveedor de servicios. Esta arquitectura distribuida permite combinar servicios de plataformas como AWS, Azure, Google Cloud o incluso infraestructura on-premise, maximizando la flexibilidad y resiliencia de los entornos tecnológicos, sin embargo, esta flexibilidad tiene un costo en la complejidad donde la gestión de la seguridad aumenta exponencialmente cuando se opera en múltiples nubes. A diferencia de un entorno unificado, donde las políticas de control y cumplimiento pueden aplicarse de forma homogénea, los entornos multi nube presentan desafíos como la falta de visibilidad consolidada, la gestión descentralizada de identidades y accesos, y la dificultad para aplicar políticas de seguridad coherentes en diferentes proveedores, cada uno con sus propios servicios, nomenclaturas, APIs y controles.

La carencia de una visión unificada sobre los activos desplegados y sus configuraciones puede derivar en configuraciones inseguras, recursos expuestos sin intención o usuarios con permisos excesivos. Además, las diferencias en los modelos de seguridad compartida entre proveedores generan incertidumbre sobre dónde termina la responsabilidad del proveedor y dónde comienza la del cliente. Estos vacíos de gobernanza representan riesgos reales que pueden ser explotados si no se gestionan proactivamente.

Para mitigar estos riesgos, han surgido soluciones específicas dentro del ecosistema de la seguridad en la nube. Entre ellas destacan:

* CSPM (Cloud Security Posture Management): soluciones diseñadas para identificar y corregir configuraciones incorrectas, detectar recursos expuestos, aplicar reglas de cumplimiento continuo (como CIS Benchmarks o NIST 800-53) y generar alertas tempranas ante desviaciones de la política de seguridad.
* CNAPP (Cloud-Native Application Protection Platform): plataformas más amplias que combinan capacidades de CSPM con análisis de vulnerabilidades en contenedores, control de acceso basado en identidad, protección en tiempo de ejecución (RASP) y escaneo de IaC. Estas herramientas permiten adoptar una visión holística de la seguridad de aplicaciones nativas en la nube.

En el marco del presente trabajo, la seguridad en entornos multi nube se aborda no solo desde una perspectiva tecnológica, sino como un desafío de diseño desde el inicio del pipeline. La combinación de herramientas como Terraform, Packer, Ansible y validadores de IaC con principios de DevSecOps permite construir una solución automatizada que incorpora controles de seguridad homogéneos desde el diseño, independientemente del proveedor utilizado. Esta estrategia responde a una necesidad crítica en entornos institucionales donde la visibilidad, el cumplimiento y la respuesta rápida ante incidentes son factores clave para la continuidad operativa y la protección de los datos.

### Herramientas y tecnologías relevantes

La implementación efectiva del enfoque DevSecOps, especialmente en entornos multi nube, exige una selección cuidadosa de herramientas que permitan automatizar tareas, versionar configuraciones, gestionar flujos de trabajo colaborativos y garantizar la seguridad desde el diseño. En este contexto, las tecnologías elegidas deben no solo facilitar la eficiencia operativa, sino también integrarse entre sí para formar un flujo coherente, auditable y seguro de provisión, configuración y despliegue de infraestructura y aplicaciones.

Entre las herramientas más relevantes utilizadas en el presente trabajo se destacan:

* **Packer:** Desarrollado por HashiCorp, Packer permite la creación automatizada de imágenes de máquina a partir de una fuente de configuración unificada. Estas imágenes pueden ser utilizadas en múltiples entornos (por ejemplo, AWS, Azure, VirtualBox o VMware) garantizando coherencia y portabilidad. Su integración con scripts de configuración y su capacidad para incorporar validaciones de seguridad lo convierten en un componente esencial del proceso de "security by design".
* **Terraform:** También de HashiCorp, es una de las herramientas más adoptadas para definir infraestructura como código (IaC). A través de su lenguaje declarativo HCL (HashiCorp Configuration Language), permite describir de forma modular, reutilizable y controlada todos los recursos de infraestructura. Su capacidad para gestionar el estado de los despliegues, validar cambios antes de aplicarlos y trabajar con múltiples proveedores (multi nube) es clave en contextos híbridos y distribuidos.
* **Ansible:** Esta herramienta de automatización de Red Hat se utiliza para la configuración post-despliegue, el aprovisionamiento de servicios y la aplicación de políticas de seguridad (hardening). Gracias a su sintaxis en YAML, es legible y accesible para equipos multidisciplinarios. Ansible facilita la aplicación repetible y controlada de configuraciones, asegurando la consistencia entre entornos y reduciendo la intervención manual.
* **Plataformas CI/CD:** GitHub y Azure DevOps: En un flujo DevSecOps maduro, la automatización de pruebas, validaciones y despliegues es imprescindible. GitHub proporciona el control de versiones del código fuente, templates IaC y configuraciones, mientras que Azure DevOps permite orquestar pipelines modulares que integran las herramientas mencionadas. Estos pipelines ejecutan validaciones sintácticas, escaneos de vulnerabilidades, pruebas unitarias y despliegues automatizados, todo bajo políticas predefinidas.

La verdadera potencia de estas herramientas se manifiesta cuando se integran en un flujo continuo, donde cada fase del ciclo de vida —desde la construcción de imágenes, la definición de infraestructura, la aplicación de configuraciones, hasta la entrega en producción— se ejecuta de forma automatizada y con controles de seguridad embebidos. Esta integración posibilita la detección temprana de errores, la trazabilidad completa de los cambios, y la garantía de que cada nuevo entorno cumple con los estándares definidos.

En el marco del presente Trabajo Fin de Máster, estas herramientas no solo se seleccionaron por su popularidad o potencia individual, sino por su complementariedad dentro de una solución que aspira a ser replicable, escalable y alineada con los principios de automatización segura y continua en entornos multi nube.

## Trabajos relacionados

El avance del enfoque DevSecOps, junto con su integración natural con prácticas de automatización e Infraestructura como Código (IaC), ha dado lugar a una creciente producción científica, desarrollos experimentales y soluciones empresariales aplicadas. Esta evolución ha motivado a comunidades académicas, centros de investigación y empresas tecnológicas a explorar nuevas formas de fortalecer la seguridad, la eficiencia y la trazabilidad en el ciclo de vida de los sistemas.

En esta sección se realiza un análisis crítico de los principales trabajos que han abordado estos desafíos desde dos perspectivas complementarias: por un lado, los aportes académicos, que se centran en la sistematización teórica, la definición de marcos conceptuales y la evaluación metodológica de herramientas y prácticas emergentes; y por otro, las soluciones comerciales e industriales, impulsadas por empresas tecnológicamente avanzadas que han implementado estos enfoques en entornos reales a gran escala.

Esta distinción permite no solo identificar el estado actual del conocimiento, sino también evidenciar las lagunas existentes entre la teoría y la práctica, permitiendo ubicar con mayor claridad la contribución del presente Trabajo Fin de Máster como una propuesta de integración técnico-práctica, orientada a ser replicable en instituciones públicas y privadas con diferentes niveles de madurez tecnológica.

En los últimos años, ha proliferado la difusión de contenido sobre IaC y DevSecOps en medios como blogs técnicos, redes sociales profesionales y plataformas de documentación comunitaria. No obstante, los trabajos académicos formales siguen siendo los que proporcionan una base más estructurada y rigurosa para comprender la evolución del paradigma. Por ejemplo, el estudio de Sharma et al. (2020) presenta un caso aplicado donde se integran herramientas como Jenkins, SonarQube y Checkmarx en un pipeline DevSecOps, logrando una mejora del 30% en la detección temprana de vulnerabilidades, y reforzando la necesidad de incorporar controles automatizados en flujos de desarrollo continuo.

De manera complementaria, Chiari et al. (2022) realizaron una revisión sistemática centrada en los riesgos de seguridad asociados al uso de scripts IaC, subrayando la urgencia de integrar validaciones semánticas y de cumplimiento en los pipelines CI/CD. Este tipo de investigaciones demuestra que el código de infraestructura también requiere auditoría, control de calidad y políticas explícitas que prevengan errores de configuración y fallos de seguridad.

En el plano empresarial, empresas como Netflix y Capital One han documentado de manera abierta sus experiencias con DevSecOps. Netflix, por ejemplo, ha diseñado sistemas internos para validación de políticas de seguridad usando Open Policy Agent (OPA) y ha utilizado herramientas como Spinnaker para orquestar despliegues seguros con controles automatizados (Netflix Tech Blog, 2020). Capital One, por su parte, ha migrado de arquitecturas tradicionales hacia entornos gestionados mediante Terraform e IaC, combinando herramientas como Trivy, Checkov y escaneos SAST/DAST para garantizar el cumplimiento normativo desde la fase de construcción de los entornos (Capital One Tech Blog, 2021).

Respecto al ámbito multicloud, Paul (2024) identifica desafíos clave como la gestión descentralizada de identidades, la poca visibilidad transversal entre entornos y la falta de políticas de seguridad homogéneas entre proveedores. Como estrategia, propone adoptar plataformas CSPM (Cloud Security Posture Management) y herramientas CNAPP (Cloud-Native Application Protection Platform) que integren análisis automático, corrección de desviaciones y cumplimiento continuo.

A pesar del avance en metodologías, herramientas y casos de éxito, aún persiste una brecha significativa en la capacidad de adopción práctica, especialmente en instituciones con recursos limitados o estructuras rígidas. Desde mi experiencia profesional y académica, he podido constatar que muchas de las soluciones más maduras requieren niveles altos de especialización técnica, licencias costosas o una infraestructura difícil de replicar en entornos educativos o gubernamentales.

Esta situación se agrava en contextos donde la formación continua es escasa y la innovación depende de iniciativas individuales más que de políticas institucionales. Por ello, resulta necesario diseñar modelos técnicos más accesibles, integrados y bien documentados, capaces de ser implementados de forma modular y progresiva. En ese sentido, cobra especial importancia el rol de espacios de aprendizaje práctico como los dojos tecnológicos, donde los equipos pueden experimentar con herramientas reales, construir flujos DevSecOps desde cero y apropiarse del conocimiento mediante la práctica.

La ausencia de estos entornos limita la transferencia de conocimiento y retrasa la adopción de prácticas modernas en organizaciones que más lo necesitan. Es por lo que este TFM se plantea no solo como una solución técnica funcional, sino como una propuesta pedagógica y operativa que pueda ser replicada, adaptada y extendida en contextos donde la automatización y la seguridad aún están en proceso de maduración. A través de una implementación basada en herramientas open source, flujos CI/CD auditables y validaciones de seguridad desde el diseño, se busca ofrecer un modelo aplicable en la realidad, útil tanto para profesionales como para formadores, y adaptable a los retos de un ecosistema tecnológico cada vez más exigente.

### Aportes académicos

**DevSecOps como marco de integración seguridad-despliegue**

Uno de los aportes más destacados en el campo académico proviene del trabajo de Myrbakken y Colomo-Palacios (2017), quienes realizaron una revisión multifocal de literatura para caracterizar el enfoque DevSecOpsn su estudio destacan que esta metodología no puede entenderse solo desde la perspectiva tecnológica, sino también como un cambio cultural y organizacional, donde la seguridad se convierte en una responsabilidad compartida entre todos los actores involucrados en el ciclo de vida del software. Esta visión rompe con el modelo tradicional de seguridad como responsabilidad exclusiva de un equipo aislado y especializado, promoviendo en su lugar un modelo de colaboración horizontal donde desarrollo, operaciones y seguridad trabajan con métricas, flujos y objetivos comunes.

Uno de los hallazgos clave de este trabajo es la necesidad de que las organizaciones desarrollen capacidades internas para integrar herramientas de seguridad (como escaneos SAST/DAST, análisis de dependencias, políticas de cumplimiento automatizadas) en sus flujos de entrega continua. Además, identifican barreras significativas para la adopción de DevSecOps, como la falta de formación especializada, la resistencia al cambio cultural, y la ausencia de una estrategia unificada de seguridad y automatización. Estas observaciones resultan especialmente relevantes en contextos institucionales como los que aborda este TFM, donde la madurez tecnológica puede ser heterogénea, y donde la integración de prácticas modernas requiere no solo herramientas, sino también una gestión del cambio bien estructurada.

En términos prácticos, la literatura sugiere que DevSecOps no debe implementarse como una capa posterior al pipeline de DevOps, sino como una serie de prácticas distribuidas a lo largo de todo el ciclo de vida. Estas incluyen desde la definición de requisitos seguros, el versionado de configuraciones sensibles, el uso de infraestructura como código validado, hasta la orquestación de despliegues con validaciones embebidas. La inclusión de herramientas como Checkov, Trivy, OPA, entre otras, permite automatizar estas tareas y liberar a los equipos de procesos manuales propensos al error.

Complementariamente, el estudio de Rajapakse et al. (2021) identifica 21 desafíos clave en la adopción de DevSecOps, a través de una revisión sistemática de literatura. Entre los principales se encuentran la dificultad para integrar herramientas dispares, la baja visibilidad de métricas de seguridad y la falta de competencias técnicas específicas en los equipos. Los autores proponen soluciones como la capacitación cruzada, la automatización de controles de seguridad y la estandarización de flujos de trabajo.

La relevancia de estos aportes para el presente TFM es directa ya que el flujo propuesto no busca solamente construir entornos automatizados, sino incorporar principios de seguridad desde la fase de diseño, apoyándose en prácticas y recomendaciones formalizadas por la literatura científica. Esta integración se manifiesta en la forma en que se orquesta el pipeline, en la elección de herramientas open source, y en la visión de que la seguridad no es un resultado final, sino una propiedad emergente del sistema bien construido, modular, auditable y replicable

**Evaluación de herramientas IaC y automatización**

La automatización de infraestructura mediante el enfoque Infraestructura como Código (IaC) ha transformado la manera en que las organizaciones gestionan sus entornos tecnológicos. Desde su adopción inicial en entornos DevOps, la IaC ha pasado de ser una práctica avanzada a convertirse en un estándar ampliamente aceptado para garantizar reproducibilidad, trazabilidad y eficiencia en la provisión de recursos. Este cambio ha sido ampliamente documentado y analizado en la literatura académica, donde se han evaluado tanto las herramientas disponibles como sus limitaciones, especialmente en lo que respecta a la seguridad, validación y mantenibilidad.

Uno de los estudios más completos en esta línea es el realizado por Özdoğan et al. (2023), quienes llevaron a cabo un análisis sistemático de las principales tecnologías de IaC, incluyendo Terraform, Ansible, CloudFormation y Puppet. El estudio clasifica estas herramientas según criterios como expresividad del lenguaje, modularidad, idempotencia, integración con pipelines CI/CD, comunidad de soporte y compatibilidad con proveedores multicloud. Sus hallazgos confirman que, si bien la IaC permite acelerar el despliegue y facilitar la estandarización de configuraciones, muchas organizaciones no aprovechan completamente su potencial, y en algunos casos replican errores estructurales al automatizar configuraciones deficientes.

En este sentido, uno de los riesgos más relevantes identificados es la falta de análisis semántico en los scripts de configuración. Aunque las herramientas actuales permiten verificar errores sintácticos mediante validadores como terraform validate o ansible-lint, no todas permiten detectar fallos lógicos o de intención. Por ejemplo, puede ocurrir que un recurso se despliegue con permisos más amplios de lo necesario, sin que eso sea una violación sintáctica del código, pero sí un problema de seguridad latente. Esta brecha entre validez sintáctica y seguridad funcional es una de las razones por las que la IaC debe integrarse dentro de un flujo DevSecOps más amplio.

Complementando esta perspectiva, Chiari et al. (2022) enfocaron su estudio en el análisis estático de scripts de IaC. A través de una revisión sistemática, identificaron que muchas organizaciones no utilizan herramientas de análisis profundo sobre su infraestructura como código, y que existe una necesidad urgente de incluir escaneos semánticos y de cumplimiento normativo directamente dentro de los pipelines de entrega continua. Herramientas como Checkov, tflint, cfn-lint o InSpec han sido diseñadas precisamente para cerrar esa brecha, permitiendo analizar no solo el código en sí, sino su impacto en términos de exposición, configuración insegura o incumplimiento de políticas.

Además, los estudios coinciden en que la adopción de IaC no se limita a escribir scripts, sino que implica cambios estructurales en la forma de trabajar. La gestión del estado (terraform.tfstate), la reutilización de módulos, la implementación de versiones controladas, y la orquestación de ejecuciones seguras en ambientes separados (dev, test, prod) son aspectos que requieren tanto formación como madurez organizacional. Por esta razón, varios trabajos destacan la necesidad de metodologías educativas, guías de estilo y repositorios compartidos que ayuden a equipos diversos a aplicar correctamente estas herramientas.

Desde la perspectiva de este TFM, los aportes académicos en torno a la evaluación de herramientas IaC permiten justificar tanto la selección de tecnologías como las medidas adoptadas para mitigar sus limitaciones. La integración de validadores estáticos (tfsec, Checkov), herramientas de escaneo de seguridad (Trivy) y pruebas funcionales dentro del pipeline CI/CD responde directamente a los hallazgos señalados por estos estudios. Además, se adopta un enfoque modular y controlado del código IaC, alineado con las recomendaciones de la literatura, para garantizar que la infraestructura generada sea no solo reproducible, sino también segura, portable y fácil de auditar.

**Seguridad en multi nube**

La adopción de arquitecturas multi nube ha ganado impulso en la última década como estrategia para evitar la dependencia de un único proveedor, mejorar la resiliencia operativa y optimizar recursos según las fortalezas de cada plataforma. Esta estrategia, que combina servicios de nubes públicas como AWS, Azure, Google Cloud y entornos on-premise, proporciona una gran flexibilidad, pero también introduce complejidades considerables en la gestión de la seguridad y el cumplimiento normativo. La academia ha comenzado a abordar estas complejidades desde perspectivas técnicas, arquitectónicas y organizativas, ofreciendo marcos de referencia que permiten comprender, evaluar y enfrentar los riesgos asociados.

En este sentido, el estudio de Paul (2024) resulta especialmente relevante, ya que sistematiza los principales desafíos vinculados a la seguridad en entornos multi nube. Entre ellos, destacan la gestión descentralizada de identidades, la ausencia de una visibilidad consolidada del entorno, y la inconsistencia en la aplicación de políticas de seguridad entre proveedores. Cada nube opera bajo sus propios principios de control de acceso, configuración de red, gestión de claves y auditoría. Esta diversidad obliga a los equipos a aprender múltiples tecnologías y adaptarse a reglas diferentes, lo que puede generar brechas, duplicidades y errores de configuración.

En el estudio Paúl propone como línea de acción la adopción de plataformas como CSPM (Cloud Security Posture Management), que permiten centralizar la visualización del estado de seguridad en múltiples nubes, detectar desviaciones con respecto a políticas declarativas y automatizar la remediación de configuraciones inseguras. Estas soluciones, que se basan en principios de monitoreo continuo y compliance-as-code, se complementan con otras herramientas del ecosistema CNAPP (Cloud-Native Application Protection Platform), que añaden análisis de vulnerabilidades en contenedores, funciones serverless y recursos efímeros, ampliando así el espectro de protección.

En el ámbito académico, estas propuestas están siendo evaluadas no solo desde su eficacia técnica, sino también desde su viabilidad de adopción en entornos institucionales y educativos. Investigaciones recientes han señalado que, aunque herramientas como CSPM y CNAPP ofrecen capacidades avanzadas, su implementación requiere habilidades específicas, procesos de madurez y una gobernanza que muchas organizaciones especialmente del sector público aún no han desarrollado plenamente. Por ello, desde la literatura se promueve una visión más accesible y gradual de la seguridad en entornos multi nube, donde la automatización y la validación previa al despliegue desempeñan un rol crucial.

En ese contexto, estudios como el de Alqahtani et al. (2021) proponen arquitecturas de seguridad basadas en políticas declarativas y verificables, que pueden ser embebidas directamente en los pipelines de integración y despliegue continuo. Usando herramientas como OPA (Open Policy Agent), estas arquitecturas permiten establecer reglas claras —por ejemplo, "ninguna máquina virtual puede tener una IP pública sin justificación"— y hacerlas cumplir de forma automática en cada despliegue, sin depender de auditorías manuales posteriores.

La aplicabilidad de estos enfoques en entornos educativos, gubernamentales o de baja madurez tecnológica ha sido discutida por varios autores, quienes recomiendan modelos escalables de adopción, comenzando por IaC seguro y pipelines validados, antes de incorporar herramientas comerciales más complejas. Esta recomendación encaja con el planteamiento del presente TFM, donde se propone una solución técnicamente robusta, pero construida sobre herramientas open source, auditables y compatibles con múltiples proveedores cloud.

En resumen, los aportes académicos en este eje coinciden en que la seguridad en entornos multi nube no debe resolverse exclusivamente con soluciones propietarias, sino con una arquitectura de automatización, validación y control progresivo, basada en principios de DevSecOps. Este enfoque adoptado en el TFM permite asegurar que cada entorno desplegado, sin importar el proveedor, cumpla con políticas de seguridad definidas, reproducibles y auditables desde la base del código.

**Aplicaciones educativas y entornos controlados**

Uno de los desafíos recurrentes en la adopción de prácticas DevSecOps en organizaciones de diversa índole es la falta de espacios formativos y de experimentación práctica. Aunque la mayoría de las herramientas y enfoques asociados a DevOps, IaC y seguridad automatizada están ampliamente documentados, su puesta en práctica efectiva requiere no solo conocimientos técnicos, sino también entornos de prueba seguros, flexibles y accesibles donde los equipos puedan experimentar, equivocarse, corregir y afianzar aprendizajes. En este contexto, la literatura académica ha comenzado a explorar activamente la dimensión educativa del DevSecOps, promoviendo entornos controlados como catalizadores del aprendizaje, la colaboración y la adopción progresiva.

Un ejemplo paradigmático de esta visión lo encontramos en el trabajo de Nascimento et al. (2020), quienes documentaron la implementación de un laboratorio académico completo para la enseñanza de prácticas DevOps y DevSecOps, utilizando únicamente herramientas open source. Su propuesta incluye la configuración de entornos con Jenkins, Ansible, SonarQube, Kubernetes y herramientas de análisis estático y escaneo de contenedores. A través de casos prácticos guiados, los participantes pueden construir pipelines CI/CD, validar código, aplicar políticas de seguridad automatizadas y simular escenarios de despliegue en múltiples entornos. La experiencia demuestra que es posible reproducir entornos reales de trabajo en contextos educativos, sin incurrir en altos costes ni depender de servicios propietarios.

Esta línea de trabajo ha sido reforzada por estudios posteriores que coinciden en señalar la importancia de contar con entornos donde los estudiantes y profesionales puedan ver el impacto directo de sus decisiones en tiempo real, especialmente en lo que respecta a la seguridad, configuración y aprovisionamiento de infraestructura. La retro alimentación inmediata, la visibilidad sobre las herramientas y la autonomía para ejecutar flujos completos de entrega continua permiten cerrar la brecha entre la teoría aprendida y la práctica aplicada, fortaleciendo competencias técnicas críticas para la industria actual.

Otro aporte interesante proviene de iniciativas que vinculan entornos controlados con procesos de certificación o mejora continua dentro de las organizaciones. En estas propuestas, los laboratorios no solo cumplen una función formativa, sino que también operan como plataformas de prueba de concepto, donde los equipos pueden validar nuevas herramientas, aplicar políticas de seguridad internas o entrenar procedimientos de respuesta a incidentes. Desde esta perspectiva, los entornos controlados no son únicamente didácticos, sino también estratégicos, ya que permiten reducir el riesgo de errores en entornos productivos mediante la validación previa en contextos simulados.

En el ámbito institucional y gubernamental, estos enfoques resultan especialmente pertinentes. Muchas entidades enfrentan limitaciones de presupuesto, infraestructura y talento especializado que dificultan la adopción directa de herramientas avanzadas o arquitecturas complejas. Por ello, varios autores destacan que la replicabilidad, portabilidad y bajo coste de los entornos educativos basados en herramientas open source es un factor clave para democratizar el acceso a prácticas modernas como DevSecOps, especialmente en países en vías de desarrollo o sectores tradicionalmente alejados de la innovación tecnológica.

La propuesta planteada en este Trabajo Fin de Máster se alinea completamente con esta visión. El sistema desarrollado ha sido diseñado no solo como una solución técnica viable, sino también como un modelo formativo replicable, que puede ser adaptado a contextos académicos, institucionales o de capacitación continua. Al basarse en herramientas como Packer, Terraform, Ansible y Azure DevOps, y estar completamente documentado y versionado en GitHub, el sistema permite que otros equipos puedan reutilizar, adaptar y extender la solución, integrándola en sus propios programas de formación o procesos de modernización tecnológica. Los aportes académicos en esta línea confirman que DevSecOps no debe ser visto únicamente como un modelo operativo para grandes empresas, sino también como una herramienta educativa poderosa, capaz de transformar la manera en que se enseña y se practica la seguridad, la automatización y la gestión moderna de infraestructura.

**Reflexión crítica y proyección del enfoque académico hacia la adopción práctica de DevSecOps**

La revisión de los principales aportes académicos en torno a DevSecOps y su integración con prácticas de automatización, infraestructura como código y seguridad en entornos multi nube, ha permitido construir un marco teórico robusto que sustenta el enfoque adoptado en este Trabajo Fin de Máster. Los estudios analizados no solo demuestran la evolución del paradigma DevSecOps como una necesidad estratégica, sino que también identifican barreras reales de adopción, limitaciones tecnológicas y desafíos culturales y operativos que aún enfrentan muchas organizaciones públicas y privadas hoy por hoy.

En conjunto, estos trabajos refuerzan la idea de que DevSecOps no es una simple extensión técnica del modelo DevOps, sino un cambio profundo en la manera de concebir, construir y mantener infraestructuras modernas. La integración de IaC, la automatización de validaciones de seguridad, la adopción de políticas declarativas y la incorporación de herramientas como Terraform, Ansible, Checkov o Trivy, configuran un ecosistema donde la seguridad es parte del diseño, no un parche posterior. Sin embargo, como también se ha señalado, muchas organizaciones aún operan sobre arquitecturas monolíticas o aplicaciones multicapa tradicionales, con procesos manuales, infraestructura no estandarizada y escasa trazabilidad, lo que limita gravemente su capacidad de respuesta ante los desafíos actuales de seguridad, escalabilidad y modernización.

Es precisamente en este punto donde el presente TFM adquiere un valor especial como contribución técnica y educativa. Al haber sido diseñado desde una perspectiva aplicada, modular y completamente basada en herramientas open source, la solución propuesta se convierte en un ejemplo práctico replicable, capaz de ser adaptado por organizaciones que deseen avanzar hacia la automatización segura sin comprometer sus sistemas existentes. Tanto si se parte de un entorno legacy como de una arquitectura multicapa funcional, la incorporación progresiva de prácticas como el versionado de infraestructura, la automatización del aprovisionamiento y la validación de seguridad puede lograrse mediante modelos de transición como el que aquí se expone.

Además, la revisión académica reafirma la importancia de promover entornos de formación práctica como laboratorios controlados o dojos tecnológicos donde estos flujos puedan implementarse, probarse, adaptarse y transferirse como conocimiento aplicado. En este sentido, este TFM no solo contribuye al avance técnico de DevSecOps, sino que ofrece a la comunidad académica, a centros de formación y a instituciones públicas o privadas un punto de partida documentado, probado y accesible para implementar estas prácticas de forma progresiva y contextualizada.

Así, de esta forma la experiencia aquí plasmada se presenta como una base sólida para trabajos futuros, tanto de investigación como de implementación, que busquen cerrar la brecha entre teoría y práctica en la adopción de DevSecOps, especialmente en contextos donde el cambio tecnológico aún es incipiente, pero urgente.

### Soluciones comerciales e industriales

**Netflix: Validación de políticas y despliegue continuo seguro**

Netflix ha sido pionera en la adopción de DevSecOps a gran escala. Su arquitectura incluye herramientas internas como Lemur (para gestión de certificados TLS) y Security Monkey (para detección de desviaciones en configuraciones de seguridad en la nube). Además, integra herramientas como Spinnaker para la orquestación de despliegues continuos con verificaciones previas basadas en políticas definidas con OPA (Netflix TechBlog, 2020).

Estas soluciones industriales demuestran la viabilidad del enfoque, aunque su replicación directa en entornos académicos o institucionales con recursos limitados no siempre es posible.

**Capital One: Terraform + Validaciones de cumplimiento**

Capital One ha compartido abiertamente su transición a IaC con Terraform y su enfoque DevSecOps, en el cual integran herramientas como Checkov, Trivy y escaneos SAST/DAST automatizados dentro de sus pipelines. Su enfoque se basa en el principio de "compliance-as-code", lo que les ha permitido detectar fallos de seguridad antes del despliegue y facilitar auditorías internas (Capital One Tech Blog, 2021).

**AWS y Azure: Marcos de referencia para seguridad**

Tanto Amazon Web Services (AWS) como Microsoft Azure han desarrollado marcos y herramientas que promueven la adopción segura de DevOps:

* AWS Well-Architected Framework – Security Pillar (AWS, 2023): Proporciona lineamientos para incorporar seguridad en cinco dimensiones, desde la identidad hasta la detección de incidentes.
* Azure DevOps + Microsoft Defender for Cloud: Permite integrar escaneos automáticos en pipelines, así como evaluar el cumplimiento de configuraciones contra políticas personalizadas.

Aunque estos frameworks son valiosos, están orientados al uso de servicios propietarios y pueden limitar la portabilidad. Su análisis sirve como inspiración para replicar lógicas similares en entornos abiertos, como se plantea en este TFM.

**Herramientas DevSecOps open source ampliamente adoptadas**

* Trivy (Aqua Security): Escaneo de vulnerabilidades en contenedores, dependencias y archivos IaC.
* Checkov (Bridgecrew): Validación de configuraciones Terraform y Kubernetes.
* Conftest + OPA: Definición y aplicación de políticas de seguridad declarativas.
* GitLab CI/CD: Orquestación de flujos de entrega continua con integraciones de seguridad.

Estas herramientas permiten replicar, en entornos controlados y sin coste, muchas de las funcionalidades de plataformas comerciales. Su integración progresiva en el flujo DevSecOps es uno de los principales aportes del presente trabajo.

### Análisis crítico y aportación del presente trabajo

A partir del análisis comparado entre los aportes académicos y las soluciones comerciales e industriales revisadas en los apartados anteriores, se evidencia que el enfoque DevSecOps ha alcanzado un nivel significativo de madurez conceptual y técnica. Sin embargo, también se hace evidente una brecha importante entre las propuestas teóricas y las posibilidades reales de adopción en organizaciones que aún operan con arquitecturas tradicionales, como sistemas monolíticos o aplicaciones multicapa. Muchas de las soluciones documentadas, especialmente las desarrolladas por grandes corporaciones tecnológicas, están diseñadas para entornos con alto grado de automatización previa, personal altamente cualificado, o presupuestos considerables para infraestructura y licenciamiento.

Asimismo, gran parte de las implementaciones académicas, aunque metodológicamente rigurosas tienden a centrarse en escenarios idealizados o muy acotados, que no siempre permiten transferir sus resultados a contextos más heterogéneos o institucionales. Esta distancia entre los modelos de referencia y la realidad operativa de muchas organizaciones, particularmente en el sector público o educativo, dificulta la adopción progresiva de prácticas DevSecOps sin un marco técnico accesible, modular y probado.

En este escenario, el presente Trabajo Fin de Máster propone una aportación integradora y aplicada, orientada a cubrir precisamente esa zona intermedia: una solución que parte de herramientas validadas por la academia e incorpora buenas prácticas de la industria, pero que a la vez es replicable, de bajo coste y adaptable a equipos con distinta madurez tecnológica. La propuesta no busca innovar creando una nueva herramienta, sino construir un flujo técnico funcional, seguro y documentado, que permita aplicar el enfoque DevSecOps de forma gradual en entornos reales, tanto de producción como de formación.

La aportación del trabajo se sustenta en cinco principios que lo diferencian de otras iniciativas:

**Accesibilidad:** el uso exclusivo de herramientas open source, sin dependencias comerciales, permite su adopción en organizaciones con recursos limitados.

**Reproducibilidad:** todo el flujo está versionado, automatizado y controlado por código, facilitando su replicación en distintos entornos (locales o cloud).

**Integración progresiva:** la solución puede incorporarse de manera modular a sistemas ya existentes, sin requerir un rediseño completo de la arquitectura.

**Validación técnica:** se integran herramientas de escaneo, pruebas, verificación de cumplimiento y health checks, que permiten evaluar la seguridad y consistencia en cada fase del pipeline.

**Propósito educativo y operativo:** el sistema desarrollado está diseñado para ser comprendido, modificado y extendido por equipos multidisciplinarios, facilitando su aplicación tanto en proyectos reales como en programas de formación.

Este TFM busca convertirse en un ejemplo de transición responsable hacia DevSecOps, no desde la abstracción académica ni desde el despliegue corporativo a gran escala, sino desde la realidad de equipos que necesitan soluciones concretas, seguras y evolutivas para modernizar su infraestructura tecnológica. Esta aportación se enmarca así en el vacío práctico identificado a lo largo presente del capítulo, y establece las bases para el desarrollo del sistema que se describirá a partir del capítulo 3.

## Conclusiones del estado del arte

El análisis del estado del arte revela que la automatización segura de infraestructura es un campo de alta relevancia y evolución constante. Se identifican avances notables en la incorporación de la seguridad en pipelines CI/CD, en el uso extendido de (IaC), y en la adopción de arquitecturas multi nube. No obstante, también se observa que la integración efectiva de estos enfoques en un flujo homogéneo, portable y validado sigue siendo un desafío.

Los trabajos revisados coinciden en destacar los beneficios de DevSecOps y la necesidad de una cultura colaborativa en la que el desarrollo, las operaciones y la seguridad actúen de manera coordinada. Del mismo modo, se reconoce la utilidad de herramientas como Terraform, Packer y Ansible para definir, configurar y validar entornos de manera automatizada. Sin embargo, se evidencia una carencia de propuestas prácticas que consoliden estas herramientas en un pipeline único, aplicable a distintos entornos y replicable sin dependencias comerciales.

Con este Trabajo Fin de Máster, se busca aportar una contribución práctica que sirva como puente entre la teoría y la aplicación real del enfoque DevSecOps. La propuesta se fundamenta en el uso de varias herramientas open source ampliamente validadas, integradas en una arquitectura modular, portable y diseñada para ejecutarse en entornos controlados y accesibles. A partir de la experiencia y los conocimientos adquiridos en el máster de DevOps de UNIR, se pretende demostrar que es posible integrar prácticas de seguridad automatizada en la gestión de infraestructura de forma efectiva, sin depender de tecnologías propietarias ni generar una carga adicional para los equipos. En este sentido, la contribución se alinea con las necesidades actuales del sector, al proponer un flujo técnico, seguro y adaptable a múltiples plataformas, que no solo automatiza el despliegue de infraestructura, sino que también garantiza su cumplimiento con estándares reconocidos de seguridad. De este modo, se busca cubrir un vacío práctico identificado tanto en la literatura como en los entornos reales de implementación, ofreciendo una solución aplicable y replicable para instituciones públicas, privadas, educativas o de tamaño medio que deseen mejorar su postura de seguridad sin asumir complejidades innecesarias.

# Objetivos y metodología de trabajo

## Objetivo general

Diseñar e implementar un proceso automatizado de DevSecOps que integre herramientas como Packer, Terraform y Ansible para generar entornos multi nube seguros, portables y reproducibles, con el fin de demostrar la efectividad del enfoque en la mejora de la seguridad, la eficiencia operativa y la estandarización del despliegue de infraestructura mediante pipelines CI/CD.

## Objetivos específicos

El presente Trabajo Fin de Máster se enmarca en una propuesta de tipo práctico, cuyo propósito es demostrar la efectividad de un enfoque DevSecOps automatizado en la gestión de infraestructura multi nube. Para ello, se han definido una serie de objetivos específicos que permiten traducir el objetivo general en metas operativas, alcanzables y observables, orientadas a la integración, automatización, validación y evaluación de procesos de infraestructura como código (IaC) con seguridad embebida desde el diseño.

Estos objetivos han sido planteados bajo un criterio de viabilidad técnica en entornos controlados, utilizando herramientas ampliamente adoptadas en la industria, con el fin de garantizar la replicabilidad y el valor formativo del trabajo. Además, cada uno de ellos busca generar un impacto tangible en la calidad, seguridad, eficiencia o trazabilidad del proceso de despliegue de infraestructuras tecnológicas.

* **Establecer un flujo técnico de adopción de DevSecOps** que combine construcción de imágenes, gestión de infraestructura como código y validación automatizada de seguridad, orientado a garantizar consistencia, trazabilidad y cumplimiento.
* **Diseñar un pipeline CI/CD** completo que integre herramientas como Packer, Terraform, Ansible y GitLab CI para automatizar la provisión, configuración y validación de entornos de infraestructura, demostrando su efectividad en procesos de entrega continua seguros.
* **Implementar la gestión automatizada de infraestructura como código (IaC)** en plataformas como AWS, Azure y VMware, evaluando su portabilidad, escalabilidad y reducción de errores de configuración.
* **Aplicar políticas de hardening automatizadas utilizando Ansible**, basadas en estándares reconocidos como CIS Benchmarks, con el objetivo de reforzar la seguridad base de los entornos antes de su puesta en producción.
* **Incorporar herramientas de análisis de seguridad y cumplimiento al pipeline** de automatización, midiendo su impacto en la detección temprana de vulnerabilidades y en el aseguramiento continuo.
* **Validar el proceso propuesto a través de pruebas controladas en entornos multi nube**, evaluando indicadores como el tiempo de despliegue, la cobertura de seguridad, la consistencia de la infraestructura y la reducción de la intervención manual.
* **Documentar el sistema desarrollado y su flujo de implementación**, detallando la estructura del repositorio, los scripts utilizados y las estrategias de validación, con el fin de garantizar su replicabilidad, reutilización y utilidad pedagógica en entornos educativos, institucionales o de formación técnica.

## Metodología del trabajo

La necesidad de construir una solución técnica funcional, segura, replicable y validada que permita aplicar el enfoque DevSecOps en entornos reales y heterogéneos, especialmente en organizaciones que aún operan sobre arquitecturas tradicionales. En los capítulos previos se ha expuesto la evolución del paradigma DevSecOps, los aportes teóricos y comerciales existentes, así como las brechas persistentes entre la teoría y la práctica. Este contexto justifica la adopción de una metodología orientada a la implementación progresiva y evaluada de un flujo técnico completo, sustentado en herramientas open source y buenas prácticas ampliamente reconocidas.

En coherencia con los objetivos definidos, se ha optado por una metodología estructurada en fases secuenciales y complementarias, alineadas con el ciclo de vida de la infraestructura y el software: desde el diseño y la construcción de imágenes base, pasando por el aprovisionamiento automatizado y la configuración segura, hasta la integración de validaciones en pipelines CI/CD y la evaluación del sistema en entornos multi nube. Cada fase ha sido diseñada de forma modular, permitiendo no solo su desarrollo individual, sino también su integración armónica dentro de un flujo DevSecOps automatizado y trazable.

La estrategia de trabajo combina un enfoque experimental y reproductible, basado en pruebas controladas y observación técnica, con una perspectiva pedagógica y operativa, que considera la importancia de documentar cada componente del sistema para facilitar su reutilización y adaptación en otros contextos. Se ha buscado, en todo momento, mantener un equilibrio entre la sofisticación técnica y la accesibilidad, de modo que la solución final sea útil tanto para entornos productivos como educativos.

La elección de herramientas Packer, Terraform, Ansible, Azure DevOps, entre otras responde no solo a su madurez tecnológica, sino también a su integración natural dentro del flujo DevSecOps. Asimismo, la validación de cada fase incluye métricas observables como el tiempo de despliegue, el cumplimiento de políticas de seguridad, la cobertura de pruebas, la consistencia de entornos y la reducción de intervención manual.

Esta metodología, centrada en la automatización y la seguridad desde el diseño, permite responder de forma concreta a los objetivos planteados y contribuye a cerrar la brecha entre las propuestas académicas y la realidad operativa de las organizaciones. En los apartados siguientes se describe con mayor detalle cada una de las fases que componen esta metodología y los mecanismos de evaluación utilizados para validar su efectividad.

### Fase I – Diseño del flujo DevSecOps

Esta fase inicial se enfoca en el diseño lógico y estructurado del flujo DevSecOps propuesto. A partir del análisis del estado del arte y las necesidades identificadas, se define la arquitectura funcional del sistema, determinando las herramientas, las etapas del pipeline y las relaciones entre los componentes. El objetivo es representar, de manera visual y declarativa, un flujo técnico reproducible que abarque desde la construcción de imágenes base hasta la validación del despliegue. Este diseño actúa como base para las fases subsiguientes, asegurando coherencia, trazabilidad y alineación con los objetivos generales del trabajo.

Esta fase inicial constituye el punto de partida para la construcción del sistema propuesto y tiene como finalidad diseñar el flujo lógico, estructurado y seguro que guiará toda la implementación técnica del enfoque DevSecOps. Se trata de una etapa fundamental, ya que establece los componentes, relaciones, procesos y herramientas que serán articulados en las fases posteriores. Además, permite representar gráficamente el ciclo de vida automatizado del sistema, identificando cada etapa clave desde la generación de imágenes base hasta el despliegue y validación final.

El diseño del flujo responde directamente a los principios recogidos en la revisión del estado del arte (Capítulo 2), donde se expone la importancia de integrar la seguridad desde fases tempranas del desarrollo, automatizar la provisión y configuración de entornos, y validar continuamente el cumplimiento de políticas. Asimismo, se alinea con los objetivos específicos planteados en este trabajo, al proporcionar un marco técnico claro, reproducible y auditable sobre el cual se sustentará la solución práctica.

Para representar de forma precisa y comprensible el flujo DevSecOps, se emplearon herramientas de diagramado como Draw.io y PlantUML, que permiten construir modelos visuales detallados sin depender de software propietario. Se optó por diagramas modulares que representan tanto el flujo horizontal del pipeline como la interacción vertical entre herramientas y fases. Estos diagramas han sido versionados en el repositorio del proyecto para facilitar su revisión y evolución a lo largo del desarrollo.

El flujo técnico diseñado se estructura en seis grandes etapas, que corresponden a las fases de esta metodología:

**Build** (Construcción de imagen base): mediante Packer, se genera una imagen preconfigurada del sistema operativo con dependencias esenciales y hardening básico.

**Scan** (Escaneo de imagen y configuración): se incorporan herramientas como Trivy y tfsec para validar la seguridad de la imagen y el código IaC antes del aprovisionamiento.

**Provision** (Aprovisionamiento de infraestructura): Terraform permite desplegar los recursos necesarios (redes, VMs, reglas de acceso) en plataformas multi nube.

**Configure** (Aplicación de políticas de seguridad): Ansible aplica configuraciones específicas de seguridad, gestión de usuarios, firewall, y otros ajustes basados en CIS Benchmarks.

**Deploy** (Despliegue de la aplicación): mediante scripts de shell, Azure DevOps orquesta el despliegue del artefacto (por ejemplo, un archivo .jar), su copia segura y el reinicio del servicio.

**Validate** (Pruebas y monitoreo): se ejecutan pruebas funcionales y se realizan health checks automatizados para verificar la integridad del sistema desplegado.

Cada etapa del flujo está asociada a una herramienta concreta, cuya elección ha sido guiada por su capacidad de integración, portabilidad, soporte comunitario y alineación con los principios DevSecOps. Además, se establecieron puntos de validación explícitos, que permiten asegurar que un paso no avance hasta que los requisitos del paso anterior hayan sido cumplidos (por ejemplo, que no se despliegue infraestructura si no pasa las validaciones de seguridad).

Un aspecto diferenciador del diseño es la incorporación de plantillas modulares en YAML para los pipelines, evitando tareas predefinidas y permitiendo una mayor transparencia y control sobre cada ejecución. Esto no solo facilita la trazabilidad, sino que también permite adaptar el flujo a diferentes aplicaciones, entornos y equipos, reforzando su valor como modelo replicable y formativo.

Como resultado de esta fase, se obtiene un modelo técnico validado del flujo DevSecOps, que será implementado, probado y documentado en las fases posteriores. Este diseño actúa como plano arquitectónico de la solución, y su claridad es clave para la integración efectiva de herramientas, el cumplimiento de los objetivos planteados y la demostración práctica de que es posible automatizar la seguridad y el despliegue de infraestructura de forma coherente, segura y eficiente.

### Fase II – Creación de imágenes base con Packer

En Esta fase tiene como propósito la generación automatizada de imágenes base que sirvan como punto de partida para la creación de entornos consistentes, seguros y reproducibles. En un enfoque DevSecOps, la imagen del sistema operativo no puede ser un artefacto manual o arbitrario: debe ser construida desde el código, versionada, auditada y validada antes de ser utilizada en entornos de pruebas o producción. Esta estrategia permite aplicar el principio de "security by design" desde la base del sistema, integrando configuraciones esenciales, dependencias controladas y medidas de hardening en el proceso de construcción de las imágenes.

Para ello, se ha seleccionado Packer como herramienta central. Desarrollada por HashiCorp, Packer permite automatizar la creación de imágenes de máquina para múltiples plataformas (VirtualBox, VMware, AWS AMI, Azure Image, entre otras) desde un único archivo de configuración. Su integración con scripts de provisionamiento (en Bash, Ansible, PowerShell, etc.) lo convierte en una herramienta ideal para establecer un estándar técnico y de seguridad en entornos multi nube. Además, su uso refuerza la trazabilidad de los cambios y facilita la auditoría del proceso de construcción.

En esta fase se desarrolló una plantilla .pkr.hcl que define:

* **La imagen base del sistema operativo:** se utilizó Ubuntu Server 22.04 LTS, por su estabilidad, soporte extendido y compatibilidad con herramientas DevOps.
* **La configuración de red y credenciales:** se incluyó un usuario técnico con claves SSH específicas, sin acceso root directo por seguridad.
* **Las herramientas mínimas preinstaladas:** OpenJDK 17, Docker CE, curl, net-tools, unzip y otras utilidades comunes en entornos de desarrollo y pruebas.
* **Scripts de hardening inicial:** desactivación de acceso root, configuración de UFW, eliminación de paquetes innecesarios, y refuerzo de permisos de archivos clave.

Todo el proceso fue automatizado para que la generación de la imagen se pueda ejecutar con un solo comando (packer build imagen.pkr.hcl) y sin intervención humana. Se utilizaron comandos packer fmt y packer validate para verificar la validez y la consistencia del archivo de configuración antes de cada compilación. Las salidas de logs se registraron para análisis posterior y como evidencia de trazabilidad.

La imagen generada sirve como base estándar para las fases posteriores del pipeline. Al tener una única fuente de verdad para todos los entornos, se garantiza que las máquinas desplegadas en diferentes etapas (desarrollo, pruebas, producción) compartirán la misma base, lo cual elimina muchas de las inconsistencias habituales en ambientes que se configuran de forma manual o independiente. Esta práctica contribuye a reducir la deuda técnica, minimizar errores humanos y acelerar el aprovisionamiento.

Como medida adicional, se implementaron controles de integridad mediante el cálculo de hashes SHA256 de las imágenes generadas, lo que permite verificar que los entornos creados a partir de ellas no han sido alterados. Además, estas imágenes se almacenaron localmente y se subieron a los repositorios de imágenes de los proveedores cloud (AWS y Azure), para ser utilizadas como base por Terraform en la Fase III.

Esta fase no solo aporta eficiencia técnica, sino también un refuerzo en la seguridad estructural del sistema. Como señalan autores como Chiari et al. (2022), muchas configuraciones inseguras provienen de la falta de control en los componentes base del entorno. Al automatizar y auditar la construcción de las imágenes, se puede garantizar que los entornos nacen con un nivel mínimo de cumplimiento y están alineados con los estándares institucionales o normativos, como los CIS Benchmarks.

La fase de creación de imágenes base con Packer establece las bases técnicas, operativas y de seguridad sobre las que se desplegará la infraestructura. Asegura la consistencia entre entornos, reduce la intervención manual y permite una validación anticipada de elementos críticos del sistema, habilitando así la automatización integral del ciclo DevSecOps en las fases siguientes.

### Fase III – Gestión de infraestructura con Terraform

Esta fase se centra en la automatización del aprovisionamiento de infraestructura mediante la herramienta Terraform, en el marco de un enfoque de Infraestructura como Código (IaC). Su objetivo es garantizar que todos los recursos de infraestructura —redes, máquinas virtuales, volúmenes, reglas de acceso, direcciones IP y configuraciones de conectividad— sean definidos, desplegados y gestionados de forma declarativa, controlada y auditable. Este enfoque elimina la necesidad de configuraciones manuales, reduce errores humanos y permite lograr consistencia y trazabilidad en cada despliegue.

La elección de Terraform se basa en su madurez, su capacidad para trabajar con múltiples proveedores (AWS, Azure, VMware, etc.) y su integración fluida con otras herramientas del ecosistema DevSecOps. Además, su uso permite gestionar el estado de la infraestructura de forma centralizada, facilitando la comparación entre el estado deseado y el estado real de los recursos, y aplicando solo los cambios necesarios mediante comandos como terraform plan y terraform apply.

Para esta fase se estructuraron los archivos de Terraform siguiendo una arquitectura modular. Se definieron módulos reutilizables para cada proveedor cloud, lo que permite mantener un diseño escalable, portable y adaptado a entornos híbridos. Los módulos incluyen:

* **Networking:** definición de VPC, subredes, tablas de enrutamiento, grupos de seguridad y puertos habilitados.
* **Instancias:** aprovisionamiento de máquinas virtuales en AWS (EC2), Azure (VM) o VirtualBox (para pruebas locales).
* **Almacenamiento:** configuración de volúmenes persistentes y sus puntos de montaje.
* **Acceso:** gestión de claves públicas SSH, variables de entorno y metadatos de conexión.

Todas las configuraciones fueron parametrizadas mediante archivos variables.tf y terraform.tfvars, permitiendo reutilizar el mismo módulo con distintas entradas según el entorno (desarrollo, pruebas o producción). Además, se utilizaron outputs.tf para exportar direcciones IP, rutas y recursos generados, lo que facilita su reutilización en fases posteriores del pipeline, como la ejecución remota de Ansible.

Se emplearon validadores como terraform validate, tflint y tfsec para comprobar:

* La sintaxis correcta de los archivos de configuración.
* La adherencia a buenas prácticas de codificación IaC.
* La detección anticipada de riesgos de seguridad, como puertos abiertos innecesarios o configuraciones demasiado permisivas.

Estas herramientas se integraron en el pipeline CI/CD, automatizando la validación antes de que cualquier despliegue sea aplicado. Con ello se refuerza la filosofía "shift left", llevando las comprobaciones de seguridad y cumplimiento al inicio del ciclo de vida de la infraestructura. Una vez validadas las configuraciones, el despliegue se ejecuta con el comando terraform apply, permitiendo el aprovisionamiento automatizado de todos los recursos definidos. Gracias al uso de workspaces en Terraform, fue posible mantener múltiples entornos aislados (por ejemplo, dev, test, prod), todos gestionados desde el mismo conjunto de archivos y módulos, pero con configuraciones específicas.

Además de los despliegues en cloud público, se realizaron pruebas locales utilizando el proveedor virtualbox para validar el funcionamiento completo del sistema sin incurrir en costes adicionales ni depender de servicios externos. Esta decisión permitió verificar que la arquitectura era portable, incluso en entornos sin conectividad a la nube.

Esta fase representa un paso crítico dentro del flujo DevSecOps, ya que transforma el despliegue de infraestructura tradicionalmente manual, lento y propenso a errores a un proceso automático, predecible y auditable. La IaC implementada con Terraform actúa como el núcleo técnico que conecta la imagen base generada con Packer, con las fases posteriores de configuración (Ansible) y despliegue de la aplicación.

La gestión de infraestructura con Terraform permite cumplir con los principios de estandarización, portabilidad y validación de seguridad definidos en los objetivos de este trabajo. Su uso en este TFM demuestra que es posible implementar IaC multicloud de forma efectiva, incluso en organizaciones que parten de entornos tradicionales, habilitando así un camino realista hacia la automatización segura y la transformación digital sostenible.

### Fase IV – Aplicación de hardening con Ansible

A medida que las organizaciones avanzan en la automatización de su infraestructura, asegurar que cada sistema desplegado cumpla con buenas prácticas de seguridad ya no es una opción, sino una necesidad crítica. Esta fase del proyecto se enfoca precisamente en ello: fortalecer los entornos creados con Terraform aplicando configuraciones de seguridad automatizadas, reproducibles y auditables, utilizando la herramienta Ansible.

En el marco de este Trabajo Fin de Máster, donde se despliega una aplicación web de tres capas (frontend, backend y base de datos), garantizar la seguridad desde la base es clave para proteger la comunicación entre capas, la integridad de los datos y la estabilidad de los servicios. Aquí es donde Ansible cobra protagonismo: al permitir definir tareas de configuración en lenguaje YAML fácil de leer, compartir y versionar, se convierte en una herramienta ideal para que los equipos de desarrollo, operaciones y seguridad trabajen en conjunto sobre un mismo estándar técnico.

Los playbooks desarrollados en esta fase permiten automatizar tareas como:

* Configuración del firewall (ufw) para exponer solo los puertos esenciales según el rol de cada servidor.
* Refuerzo de los permisos del sistema y desactivación de servicios innecesarios.
* Gestión de usuarios, SSH y autenticación segura.
* Aplicación de parámetros de seguridad recomendados por los CIS Benchmarks para sistemas Linux.

Estos ajustes se aplican de forma diferenciada según el tipo de nodo: por ejemplo, la capa frontend puede exponer el puerto 80/443, mientras que el backend y la base de datos solo deben comunicarse internamente. Esta lógica fue incorporada en los inventarios dinámicos generados desde los outputs de Terraform, permitiendo que Ansible reconozca qué rol cumple cada máquina y configure sus políticas de forma contextualizada.

Lo más relevante de esta fase es que no se ejecuta de forma aislada, sino como parte del pipeline automatizado de Azure DevOps, lo cual garantiza que las políticas de seguridad se apliquen cada vez que se despliega un entorno, sin depender de pasos manuales ni de “ajustes después del despliegue”. El hardening forma así parte natural del flujo de entrega continua, alineado con el principio de “security as code”.

Esta integración también facilita la trazabilidad: cada cambio en los playbooks puede ser versionado, revisado por pares y validado automáticamente. De esta manera, el conocimiento técnico no queda encapsulado en una persona o un equipo, sino que se transforma en un recurso compartido y replicable, accesible incluso para quienes están aprendiendo a aplicar prácticas DevSecOps.

Esta fase representa un paso clave hacia la madurez operativa y la seguridad proactiva del sistema. Ansible no solo ayuda a construir servidores más seguros, sino que permite hacerlo de forma automática, estandarizada y adaptable a diferentes realidades organizacionales. Este enfoque, unido a la potencia de Azure DevOps como orquestador del pipeline, demuestra que es posible aplicar prácticas de seguridad avanzadas incluso en proyectos sencillos, sin necesidad de grandes inversiones ni plataformas propietarias.

### Fase V – Orquestación en pipeline CI/CD

La integración de todas las fases anteriores en un flujo automatizado marca un punto de inflexión en la metodología DevSecOps. Esta fase tiene como objetivo construir un pipeline CI/CD modular y seguro en Azure DevOps, que orqueste de forma coherente y trazable las actividades de construcción de imágenes, validación de código, aprovisionamiento de infraestructura, configuración segura y despliegue de servicios para una aplicación web de tres capas: frontend, backend y base de datos.

Azure DevOps fue elegido plataforma principal de orquestación por su capacidad de integrar repositorios Git, gestionar pipelines YAML altamente personalizables, y facilitar la implementación de flujos de trabajo seguros, con integración de validaciones, políticas de aprobación y seguimiento detallado. Esta decisión también responde a su compatibilidad con entornos híbridos y multi nube, así como a su facilidad de adopción institucional en entornos de formación o transición tecnológica.

El pipeline se estructura en fases secuenciales y desacopladas, siguiendo el principio de ejecución progresiva con validaciones intermedias. Cada fase cumple una función específica dentro del ciclo de vida de la aplicación y contribuye a la automatización completa del proceso. A continuación, se describe cada una de estas fases y su propósito dentro del flujo:

**El flujo CI/CD se estructura en cinco etapas clave:**

**build: Compilación automatizada del backend**

Esta primera etapa se encarga de compilar el backend desarrollado en Spring Boot. Para ello, se ejecuta un script Bash personalizado que invoca ./mvnw clean install, generando el artefacto .jar a partir del código fuente. La compilación se ejecuta dentro de un contenedor limpio construido previamente con Packer, lo cual garantiza un entorno homogéneo, seguro y replicable en cada ejecución del pipeline.

No se utilizan tareas preconfiguradas del asistente de Azure DevOps. En su lugar, se emplea un bloque de ejecución bash que encapsula toda la lógica de compilación, permitiendo mayor control y adaptabilidad. El archivo resultante se almacena temporalmente en el agente para ser utilizado en fases posteriores.

**Sast: Análisis estático de código fuente y dependencias**

Una vez generado el artefacto, se ejecutan análisis de seguridad y calidad sobre el código fuente y sus dependencias. Esta etapa cumple la función de incorporar controles DevSecOps dentro del pipeline, bajo el principio de "shift left security".

Se integran las siguientes herramientas de análisis:

SpotBugs: inspección de bytecode Java para detectar errores frecuentes y patrones de vulnerabilidad

**Checkstyle: verificación de cumplimiento de estándares de codificación.**

Trivy: escaneo de vulnerabilidades en las dependencias del proyecto definidas en pom.xml.

La ejecución de estos análisis también se realiza mediante scripts Bash, lo que permite mantener independencia del sistema de orquestación. El pipeline se configura para detenerse automáticamente si se detectan errores críticos o vulnerabilidades de alta severidad no justificadas.

**Release: Publicación del artefacto compilado**

Superada la etapa de análisis, el artefacto .jar es publicado en un repositorio accesible por el pipeline para su posterior despliegue. En este caso, se puede utilizar GitHub Packages o un repositorio privado configurado con autenticación mediante token de acceso.

El proceso se realiza mediante un paso bash que ejecuta el envío del archivo utilizando curl, scp o gh según la estrategia adoptada. Esta acción garantiza que el artefacto esté disponible para las fases de despliegue sin necesidad de mantenerlo localmente en el agente.

La fase release también incluye la generación de un identificador único de versión (por timestamp o hash Git), que será referenciado en los logs y registros del sistema, facilitando el seguimiento del ciclo de vida de cada versión.

**Provision: Aprovisionamiento automatizado de la infraestructura**

A continuación, el pipeline ejecuta la fase de provision, en la que se aplica el módulo de Terraform correspondiente para crear, actualizar o validar el entorno de ejecución. Este módulo ha sido desarrollado de forma modular y parametrizada, permitiendo gestionar:

* La red virtual y subredes específicas para frontend, backend y base de datos.
* Las máquinas virtuales necesarias en el entorno (Ubuntu Server).
* Las reglas de acceso y seguridad por grupo funcional (SG/NASG).
* La asociación de claves públicas para el acceso remoto seguro.

El estado de Terraform se mantiene remoto y versionado. Antes de aplicar cambios, se ejecuta terraform plan y se solicita validación manual si se trata de entornos críticos. El despliegue se realiza automáticamente mediante terraform apply sin interacción adicional del usuario.

**Deploy: Despliegue de la aplicación en entornos destino**

La última etapa consiste en desplegar el artefacto sobre la infraestructura provisionada. Este paso también se realiza mediante scripts Bash definidos en el pipeline YAML y sigue una secuencia clara:

* Transferencia del artefacto al servidor backend mediante SCP sobre SSH.
* Respaldo de la versión anterior, si existe, para permitir rollback controlado.
* Actualización del sistema: el nuevo .jar es copiado a la ubicación del servicio.
* Reinicio del servicio mediante systemctl restart para activar la nueva versión.
* Health check HTTP a la URL /api/status para validar la correcta disponibilidad del backend.

Para el frontend, los archivos estáticos HTML y CSS son copiados al servidor web (/var/www/html), mientras que el servidor de base de datos PostgreSQL, ya configurado mediante Ansible, se valida con una consulta SELECT 1.

La siguiente tabla resume las fases principales del pipeline CI/CD diseñado para este trabajo, especificando las herramientas utilizadas, las tareas ejecutadas y los criterios de validación aplicados.

|  |  |  |  |
| --- | --- | --- | --- |
| **Fase** | **Herramienta/s** | **Tarea principal** | **Validación** |
| Build | Docker, Bash, Maven | Compilación de código backend (.jar) y preparación de artefactos | Ejecución de scripts sin errores, artefacto generado |
| SAST | SpotBugs, Trivy, tfsec | Análisis estático de código y dependencias, detección de vulnerabilidades | Resultados sin errores críticos o CVEs de alto riesgo |
| Release | Azure DevOps, GitHub Packages | Publicación de artefactos generados en repositorio accesible | Artefacto disponible y accesible en el repositorio |
| Provision | Terraform | Aprovisionamiento de infraestructura según plantillas definidas | Infraestructura creada correctamente y con estado trazado |
| Deploy | SCP, systemctl, psql | Transferencia del artefacto al servidor, despliegue y reinicio del servicio | Aplicación funcionando, configuración segura aplicada |
| Validate | Curl, systemctl, logs, psql | Validación de endpoints, estado de servicios, conectividad y logs | Servicios activos, respuesta HTTP 200, sin errores críticos en logs |

Tabla 3 - Fases del pipeline CI/CD con herramientas aplicadas Fuente: Elaboración propia

Para facilitar la comprensión visual del proceso de automatización implementado, se incluye un diagrama de flujo que representa las fases del pipeline CI/CD orquestado en Azure DevOps. Este gráfico resume el recorrido completo de integración y despliegue continúo aplicado a la aplicación web desarrollada, reforzando la trazabilidad y coherencia de cada etapa.

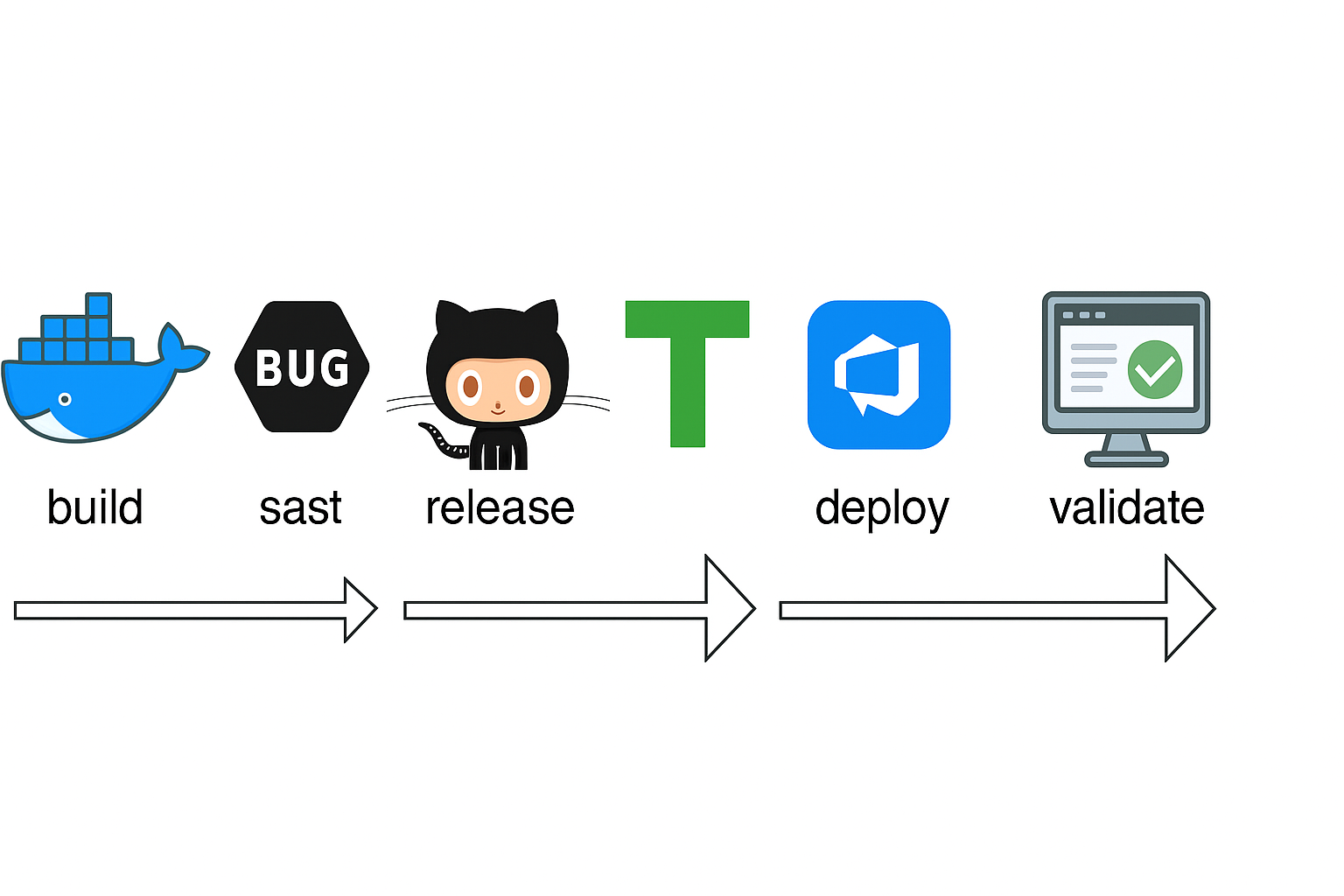


Ilustración 1 - Flujo del pipeline CI/CD Fuente: Elaboración propia

### Fase VI – Despliegue continuo y validación técnica.

La validación del sistema desarrollado constituye una fase determinante para demostrar el valor práctico de la propuesta presentada en este trabajo. A diferencia de enfoques teóricos o simulados, esta solución ha sido completamente implementada, probada y ejecutada exitosamente en un entorno real, reproduciendo de forma controlada un escenario de despliegue automatizado para una aplicación web de tres capas. El sistema desarrollado no solo ha cumplido con los objetivos funcionales y técnicos establecidos, sino que ha servido como una respuesta concreta a un problema común en muchas organizaciones: la ausencia de documentación clara, modular y ejecutable para implementar pipelines DevSecOps efectivos y seguros.

La solución ha sido estructurada en fases dentro de un pipeline CI/CD en Azure DevOps, y su validación operativa se ha realizado sobre una infraestructura real generada con Terraform, configurada con Ansible y gestionada con scripts personalizados. Las pruebas confirman que la orquestación del proceso de compilación, análisis, despliegue y validación es completamente funcional, autónoma y repetible.

Los principales aspectos validados incluyen:

Ejecución continua del pipeline sin intervención manual, con éxito en las fases build, sast, release, provision y deploy.

Despliegue completo de la solución desde cero, incluyendo la provisión de la infraestructura, configuración del entorno y publicación de la aplicación web, con sus tres capas funcionales (frontend estático, backend Spring Boot y base de datos PostgreSQL).

Validación funcional de los servicios, mediante pruebas HTTP (curl) a los endpoints del frontend y backend, y conexiones a la base de datos (psql) para asegurar la disponibilidad y operatividad del sistema.

Control de servicios en los servidores, con revisión de estado mediante systemctl, análisis de logs (journalctl) y verificación de puertos abiertos, lo que permitió confirmar la correcta aplicación de configuraciones.

Evaluación de seguridad integrada, con herramientas como tfsec, Trivy y Checkov ejecutadas automáticamente como parte de la fase sast, lo cual permitió detectar y corregir posibles vulnerabilidades de código, dependencias y configuraciones de infraestructura.

Además de la verificación técnica, se ha registrado información cuantitativa sobre los tiempos de ejecución del pipeline, el tiempo promedio por fase, y los beneficios obtenidos frente a un escenario manual de despliegue. Los resultados muestran una reducción significativa en los errores humanos, mayor velocidad de entrega y mejoras sustanciales en la trazabilidad del proceso.

Otro aspecto relevante validado en esta fase es la portabilidad y aplicabilidad institucional de la solución. Al estar basada en herramientas open source, scripts Bash y archivos YAML versionados, el sistema puede ser replicado fácilmente por equipos con conocimientos básicos en DevOps, incluso en organizaciones que aún trabajan con arquitecturas monolíticas o ambientes tradicionales. Este trabajo, por tanto, no solo aporta una solución técnica funcional, sino también una guía de implementación clara y reutilizable, que suple la falta de documentación práctica existente en el ámbito de la automatización segura en entornos híbridos o multicloud.

La validación técnica ha confirmado que el sistema desarrollado funciona de forma autónoma, segura y coherente con los principios DevSecOps, resolviendo no solo los desafíos tecnológicos, sino también una de las barreras más frecuentes en su adopción: la escasez de ejemplos reales, completos y técnicamente documentados que permitan replicar este tipo de procesos con confianza y sin depender de soluciones propietarias.

Durante la implementación de la solución propuesta, se llevaron a cabo distintas validaciones técnicas orientadas a garantizar la calidad, seguridad y funcionalidad de cada componente del sistema. La siguiente tabla resume estas validaciones, indicando el tipo de verificación aplicada, las herramientas utilizadas y los resultados obtenidos.

|  |  |  |  |
| --- | --- | --- | --- |
| **Componente** | **Validación realizada** | **Herramienta** | **Resultado** |
| Código fuente | Revisión de estilo y errores estáticos con SpotBugs | SpotBugs | Sin errores críticos de compilación o estilo |
| Dependencias y artefactos | Escaneo de vulnerabilidades con Trivy | Trivy | Sin CVEs de severidad alta |
| Infraestructura | Validación de plantillas IaC con tfsec y terraform validate | tfsec, terraform | Plantillas aprobadas y seguras |
| Servicios desplegados | Health check de APIs y servicios (HTTP, logs, systemctl) | curl, systemctl, logs | Servicios activos, sin errores en logs |
| Base de datos | Pruebas de conectividad y consultas básicas con psql | psql | Base de datos accesible y funcional |

Tabla 4 - Validación del sistema Fuente: Elaboración propia

### Fase VII – Conclusión técnica del desarrollo

La implementación completa del pipeline DevSecOps automatizado para el despliegue de una aplicación web de tres capas ha permitido demostrar, de manera tangible, que es posible construir soluciones robustas, seguras y adaptables sin depender de plataformas propietarias ni flujos complejos. Esta experiencia práctica ha cerrado el ciclo iniciado en la fase de diseño, validando que una arquitectura construida bajo principios de seguridad, automatización e infraestructura como código puede ser ejecutada exitosamente en un entorno real y controlado.

Uno de los principales logros de este desarrollo radica en la modularidad y claridad técnica de cada componente, lo cual facilita su entendimiento, mantenimiento y extensión por parte de otros equipos técnicos. Desde la compilación automatizada del backend con Spring Boot hasta la provisión de infraestructura con Terraform y la aplicación de hardening con Ansible, cada fase del pipeline ha sido construida de forma cohesionada, documentada y funcional. La elección de Azure DevOps como plataforma de orquestación ha reforzado la trazabilidad del proceso y ha facilitado la integración entre etapas.

Además de cumplir con los objetivos funcionales y técnicos planteados, este trabajo aporta un valor adicional: sirve como ejemplo práctico y reutilizable para instituciones, equipos de TI o entornos educativos que buscan adoptar metodologías DevSecOps en sistemas existentes o en nuevos desarrollos. En particular, la documentación generada, el uso exclusivo de herramientas open source y la estructuración del flujo mediante scripts y YAMLs reutilizables permiten que esta solución sea replicada por organizaciones con diferentes niveles de madurez tecnológica, incluyendo aquellas que aún operan sobre arquitecturas monolíticas o multicapa tradicionales.

Otro aspecto destacable es la forma en que esta implementación aborda una brecha real en el ecosistema DevOps: la falta de ejemplos prácticos integrales, documentados y funcionales que combinen seguridad, automatización y portabilidad. Este TFM no solo propone un modelo conceptual, sino que ofrece una solución concreta, validada y respaldada por código funcional disponible públicamente, lo cual le otorga un valor operativo inmediato para quienes enfrentan los desafíos de migrar hacia infraestructuras más ágiles y seguras.

Con esta fase final se confirma que la solución desarrollada no solo funciona correctamente, sino que está lista para ser adoptada, adaptada y extendida en contextos reales. Su diseño práctico, modular y pedagógico permite que trascienda el alcance del TFM como ejercicio académico, convirtiéndose en un recurso útil para la formación continua de profesionales, la estandarización de procesos internos en organizaciones, y el fortalecimiento de capacidades técnicas en torno a la automatización y la seguridad.

# Desarrollo específico de la contribución

En este capítulo se detalla el proceso completo de diseño, implementación y evaluación técnica de la solución práctica propuesta en este Trabajo Fin de Máster. La contribución se enmarca en un enfoque de desarrollo práctico, orientado a demostrar la viabilidad de integrar varias herramientas en un flujo DevSecOps automatizado, adaptable a entornos multicloud y con políticas de seguridad embebidas desde el inicio.

Para lograrlo, se siguió una metodología estructurada que abarca desde la planificación y análisis de requisitos, hasta la construcción de un pipeline funcional y su validación en entornos controlados. El objetivo no ha sido únicamente desarrollar una solución técnica funcional, sino justificar las decisiones adoptadas durante el proceso, documentar su aplicabilidad y valorar su impacto en términos de eficiencia, seguridad y portabilidad.

A lo largo de este capítulo se presentan los siguientes subapartados: en primer lugar, se expone la planificación, el análisis de requisitos y el diseño general del sistema; a continuación, se describe con detalle la implementación técnica del producto desarrollado; posteriormente, se realiza una evaluación estructurada basada en pruebas reales; y finalmente, se analizan las limitaciones del sistema y se formula una conclusión técnica que resume los principales aprendizajes del proceso, el código donde se demuestra el flujo completo esta versionado en GitHub en la siguiente dirección <https://github.com/jorgearmijo882/DevOps_TFM.git>

## Planificación / Análisis / Requisitos

Desde una perspectiva técnica y operativa, abordar un proyecto como este implica mucho más que ensamblar herramientas; requiere comprender el contexto real en el que se pretende aplicar la solución. Como profesional con experiencia en entornos on-premise y cloud, he tenido la oportunidad de observar los retos que enfrentan las instituciones al tratar de incorporar seguridad, automatización y consistencia en sus procesos de infraestructura. Esta sección recoge el trabajo previo que permitió guiar el desarrollo, estructurar el flujo técnico y definir los requisitos funcionales y no funcionales del sistema.

### Identificación del problema y contexto de uso

En muchos entornos, especialmente aquellos de naturaleza pública o institucional, los procesos de despliegue de infraestructura siguen dependiendo en gran medida de procedimientos manuales, no trazables, y sin controles de seguridad integrados desde el origen. Esta situación genera entornos inconsistentes, propensos a errores, difíciles de replicar y costosos de auditar.

Desde mi experiencia como administrador de sistemas y consultor técnico, he visto cómo estas deficiencias impactan negativamente en la agilidad operativa y en la postura de seguridad de las instituciones. El problema no radica solo en la falta de herramientas, sino en la ausencia de una estrategia coherente de automatización segura.

El desarrollo de este TFM parte, por tanto, de una necesidad concreta de construir un flujo DevSecOps aplicable, reproducible y accesible, que permita a instituciones públicas o privadas con recursos limitados adoptar buenas prácticas sin depender de soluciones propietarias. Así también el presento documento pretende ser un referente académico para quien se inicializa en el mundo de DevOps, la adopción de Infraestructura como código y el uso de nubes públicas. Se considera como entorno de uso una organización con equipos mixtos, servicios en la nube (AWS, Azure) y estructuras legacy on-premise, que requieren una transición gradual hacia una infraestructura gestionada como código y validada desde su creación.

### Tecnologías seleccionadas y justificación

Uno de los aspectos más relevantes al iniciar el proyecto fue elegir herramientas que no solo fueran potentes, sino también estables, abiertas y con una curva de aprendizaje razonable. La selección tecnológica respondió a criterios como la compatibilidad entre plataformas, el soporte comunitario, la capacidad de automatización y la transparencia operativa.

* **Packer** fue la elección natural para la construcción de imágenes base. Su modelo declarativo y su compatibilidad con múltiples backends permite garantizar consistencia en los entornos, desde entornos virtualizados hasta nubes públicas.
* **Terraform** aporta una manera elegante y controlada de definir infraestructura como código. Su lenguaje HCL, la gestión de estado y modularidad encajan perfectamente con los principios de diseño reutilizable y portable.
* **Ansible** permite aplicar configuraciones seguras de forma idempotente y clara. Como administrador de sistemas, valoro especialmente su legibilidad y la facilidad con la que se puede mantener un inventario y playbooks reutilizables.
* **GitLab CI/CD (Community Edition)** fue clave para orquestar todo el flujo. Como desarrollador, reconozco el valor de contar con pipelines definidos por YAML, que integran etapas de validación, despliegue y control con total trazabilidad.

### Organización del desarrollo

Organizar un proyecto que involucra tantas capas de infraestructura, automatización y seguridad requiere no solo una visión técnica, sino también una planificación clara y adaptable. El desarrollo se dividió en seis fases estructuradas de forma iterativa e incremental:

1. **Modelado del flujo DevSecOps**: identificación de etapas clave, herramientas necesarias y lógica de integración.
2. **Creación de imágenes base seguras con Packer**: definición de plantillas y validación por línea base.
3. **Despliegue modular con Terraform**: configuración de infraestructura desde archivos declarativos con estado gestionado localmente.
4. **Aplicación de hardening con Ansible**: ejecución de políticas de seguridad predefinidas sobre entornos provisionados.
5. **Escaneos y validaciones de seguridad**: integración de herramientas para asegurar cumplimiento antes del uso de entornos.
6. **Automatización del pipeline CI/CD**: definición de flujo y validaciones de la aplicación de ejemplo desarrollada.

Este enfoque me ha permitido avanzar con la validación en cada componente por separado, corrigiendo desviaciones a tiempo y asegurando compatibilidad entre las fases. La documentación fue mantenida de forma progresiva, asegurando que cada decisión técnica estuviera respaldada por pruebas funcionales o escaneos.

La siguiente tabla resume la planificación temporal del desarrollo práctico, estructurado en fases secuenciales que permitieron abordar de forma progresiva el diseño, construcción, validación y documentación del sistema propuesto. Este cronograma no solo refleja el orden lógico de ejecución, sino que también evidencia la organización técnica adoptada, en coherencia con el enfoque iterativo y experimental que guía la metodología aplicada.

|  |  |  |  |
| --- | --- | --- | --- |
| **Fase** | **Actividad Principal** | **Herramientas Utilizadas** | **Semana(s)** |
| Fase I | Análisis del problema, identificación de requisitos | - | 1 – 2 |
| Fase II | Selección y configuración de herramientas | Terraform, Ansible, Azure DevOps | 3 – 4 |
| Fase III | Desarrollo de scripts y módulos IaC | Bash, YAML, Docker | 5 – 6 |
| Fase IV | Desarrollo y prueba del pipeline CI/CD | Azure DevOps Pipelines | 7 – 8 |
| Fase V | Validación técnica y ajustes finales | tfsec, Trivy, curl | 9 – 10 |
| Fase VI | Evaluación, documentación y presentación | Word, PlantUML | 11 – 12 |

Tabla 5 - Cronograma del desarrollo Fuente: Elaboración propia

### Técnicas utilizadas

Este trabajo ha sido desarrollado de forma unipersonal, lo que ha implicado asumir de manera autónoma tanto la investigación como la planificación y la ejecución técnica del sistema. Aunque no cuento con una experiencia extensa en desarrollo de aplicaciones, mi formación académica y práctica en la vida profesional me han permitido adquirir competencias sólidas en el uso de herramientas de validación de código, automatización de infraestructura y administración de plataformas tanto on-premise como en la nube, a un nivel técnico que ha permitido plantear esta alternativa practica para comprender el flujo de DevSecOps aplicado para multi nubes. Durante el proceso, hice uso de recursos ampliamente documentados por la comunidad, guías oficiales y referencias académicas que orientaron muchas de las decisiones tomadas. A pesar de no haber contado con la participación directa de un equipo de expertos en el área de desarrollo, se aplicaron prácticas profesionales propias del entorno DevOps, como:

* Control de versiones y ramificación con Git, simulando entornos colaborativos.
* Validación estática y dinámica de código con herramientas como ansible-lint, tflint, yamllint, terraform validate y packer validate.
* Revisión y comparación de configuraciones con estándares reconocidos como CIS Benchmarks y recomendaciones del NIST.
* Pruebas funcionales sobre entornos simulados para evaluar el comportamiento de los despliegues y la ejecución de políticas de seguridad.

### Instrumentos de seguimiento y evaluación

Para garantizar el control técnico del desarrollo y la calidad de la solución propuesta, se aplicaron mecanismos de seguimiento estructurados que permitieron validar los avances en cada fase del pipeline DevSecOps. Estos instrumentos reflejan prácticas habituales en entornos profesionales, orientadas a mantener la trazabilidad, reducir el error humano y facilitar la mejora iterativa.

El proceso se apoyó principalmente en Git como sistema de control de versiones, organizando el repositorio por ramas específicas asociadas a cada componente (Packer, Terraform, Ansible). Esta estrategia facilitó pruebas independientes, validación modular y documentación del histórico de cambios.

## Descripción del sistema desarrollado / Implementación

El sistema desarrollado representa una solución DevSecOps integral orientada a la gestión segura, automatizada y portable de una aplicación multicapa. Esta aplicación incluye un frontend estático, un backend basado en Java y una base de datos relacional, todos ellos orquestados mediante contenedores Docker y gestionados como infraestructura como código. A esto se suma un flujo de CI/CD modular desplegado en Azure DevOps, y un sistema de infraestructura como código (IaC) construido con Terraform y Packer, lo cual permite garantizar consistencia desde la creación de las imágenes base hasta el aprovisionamiento en la nube (AWS y Azure). El código fuente de los componentes se aloja en GitHub, desde donde se integran los flujos de integración y despliegue continuo mediante Azure DevOps, facilitando una implementación modular, auditable y adaptable a entornos híbridos en AWS y Azure.

El esquema de redes y dependencias ha sido configurado para que las comunicaciones internas se mantengan seguras, separando responsabilidades por servicio y reforzando la trazabilidad de errores.

El sistema implementado en este Trabajo Fin de Máster corresponde al desarrollo e integración de un flujo DevSecOps completo, aplicado al ciclo de vida de una aplicación web de arquitectura multicapa. El objetivo principal fue demostrar cómo la automatización, la infraestructura como código y la seguridad continua pueden integrarse en un pipeline reproducible, escalable y funcional sobre entornos híbridos y multicloud

A continuación, una ilustración del flujo del sistema:

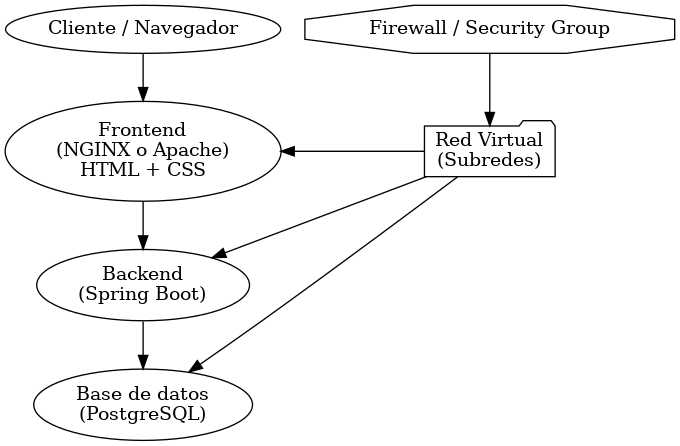


Ilustración 2 - Diagrama de arquitectura lógica del sistema Fuente: Elaboración propia

### Arquitectura multicapa y contenedores

La aplicación se estructura en tres capas desacopladas:

* Frontend: HTML/CSS/JavaScript desplegado mediante un contenedor NGINX.
* Backend: Servicio REST desarrollado en Java (Spring Boot), expuesto como un contenedor Docker basado en imagen personalizada.
* Base de datos: PostgreSQL, ejecutada como servicio contenedor con persistencia en volumen.

Cada componente ha sido encapsulado en un contenedor Docker independiente, definido mediante Dockerfile y orquestado mediante docker-compose.yml, lo que facilita su despliegue en diferentes entornos y la ejecución de pruebas de integración local.

El esquema de redes y dependencias ha sido configurado para que las comunicaciones internas se mantengan seguras, separando responsabilidades por servicio y reforzando la trazabilidad de errores.

El diseño permite representar entornos reales de instituciones que operan con aplicaciones heredadas o multicapa, y que buscan migrar hacia modelos más automatizados sin depender exclusivamente de arquitecturas basadas en microservicios. La modularidad definida permite reemplazar componentes sin afectar el resto del sistema, habilitando una evolución progresiva de cada capa.

### Construcción de imagen base con Packer

Como parte del enfoque DevSecOps, se utilizó Packer para construir una imagen base segura y reproducible que sirviera como plantilla para las instancias remotas (servidores de despliegue). Esta imagen incluye:

* Sistema base optimizado (Ubuntu Server 22.04)
* Usuario técnico con permisos mínimos y clave SSH
* Java OpenJDK, Docker, y dependencias necesarias
* Herramientas de monitoreo básico
* Reglas de hardening básico (como desactivación de root remoto)

El proceso de construcción fue automatizado mediante un archivo imagen.pkr.hcl y ejecutado dentro del pipeline, asegurando que todos los entornos desplegados sean consistentes y auditables desde su creación.

### Provisionamiento con Terraform

Se integró Terraform para gestionar la infraestructura como código en AWS y Azure. Los módulos definidos permiten:

* Crear redes virtuales (VPC/Subnets o VNets)
* Provisionar instancias EC2 o máquinas virtuales Azure
* Configurar reglas de seguridad (puertos abiertos solo a lo necesario)
* Asociar IPs públicas, claves SSH, y recursos de almacenamiento

Cada entorno (desarrollo, pruebas y producción) puede definirse como un workspace diferente en Terraform, lo que refuerza la segregación y portabilidad. Los archivos .tf están organizados modularmente por proveedor, y se gestionan desde el mismo pipeline CI/CD para un control unificado.

### Construcción de imagen Docker para compilación Java

Para garantizar la coherencia en los entornos de compilación, se desarrolló una imagen Docker personalizada enfocada a la construcción segura y eficiente del backend Java. Esta imagen incluye:

* Java OpenJDK 17
* Maven preinstalado
* Herramientas de análisis estático como spotbugs y checkstyle
* Soporte para ejecución en pipelines CI
* Configuración mínima de red y sistema

Este contenedor permite compilar el .jar de forma eficiente y reproducible tanto en local como dentro del pipeline CI/CD, sin exponer herramientas innecesarias en producción. El Dockerfile ha sido optimizado para minimizar la superficie de ataque, utilizando imágenes base oficiales y reduciendo la capa de comandos. El objetivo es asegurar que cualquier desarrollador o agente de CI pueda compilar la aplicación de forma reproducible y segura, integrando validaciones y escaneos desde el momento de la construcción.

### Pipeline CI/CD modular en Azure DevOps

Se implementó un pipeline CI/CD en Azure DevOps totalmente modular, compuesto por plantillas YAML reutilizables y parametrizadas. La estructura del pipeline se divide en los siguientes stages principales:

* **build:** Compila la aplicación backend utilizando la imagen Docker de compilación previamente generada. Toda la lógica de construcción (mvn clean install) se realiza mediante scripts bash, sin tareas predefinidas.
* **sast:** Ejecuta análisis estático de seguridad y calidad sobre el código fuente. Se integran herramientas como spotbugs, checkstyle y trivy para revisar dependencias y configuraciones.
* **release:** Publica el artefacto generado (.jar) en un sistema de almacenamiento de artefactos (por ejemplo, GitHub Packages o un repositorio privado accesible por token).
* **provision:** Ejecuta Terraform para crear o actualizar la infraestructura necesaria para el entorno de despliegue.
* **deploy:** Transfiere el artefacto al servidor remoto, respalda la versión anterior, reinicia la aplicación y verifica el estado con un health check HTTP.

Cada uno de estos stages se ha definido en templates YAML independientes, permitiendo su reutilización en diferentes proyectos. La parametrización de rutas, variables y comandos permite adaptar los flujos sin duplicar lógica.

El diseño modular del pipeline facilita la extensión con nuevos stages, su mantenimiento y la trazabilidad de ejecución.

### Despliegue continúo automatizado

El pipeline incluye un stage final deploy que realiza el despliegue automatizado de la aplicación en un servidor remoto Linux (hosted en AWS o Azure). Este proceso no depende de tareas predefinidas de Azure DevOps y se realiza completamente mediante scripts shell personalizados. Las acciones ejecutadas son:

* Descarga del artefacto .jar desde el sistema de artefactos publicado.
* Conexión segura al servidor remoto en EC2 o Azure VM (mediante SSH con claves gestionadas).
* Respaldar la versión actual del artefacto y los logs.
* Reemplazar el artefacto antiguo con el nuevo, reiniciando el servicio backend (por ejemplo, mediante systemctl restart app-backend.service).
* Ejecución de health check HTTP (curl a /actuator/health) para asegurar la disponibilidad del servicio tras el despliegue.

Se utilizaron scripts shell en lugar de tareas predefinidas para asegurar portabilidad y control del proceso. Este stage permite controlar el comportamiento del sistema incluso tras su puesta en marcha.

### Compatibilidad multicloud en AWS y Azure

El sistema fue validado para ser ejecutado en servidores Linux alojados tanto en AWS (EC2 Ubuntu con permisos limitados) como en Azure (máquinas virtuales administradas por IP pública y claves SSH). La modularidad de la solución permite:

* Reutilizar la misma configuración de Docker y base de datos en ambas plataformas.
* Aplicar el pipeline sin modificar los scripts, utilizando variables para IP, puertos y credenciales.
* Integrar validaciones y reportes adaptados a la nube destino (por ejemplo, alerta de puertos abiertos en AWS con Security Hub).

El uso de herramientas libres y estándar asegura que el sistema puede ser replicado en entornos híbridos sin requerir dependencias propietarias.

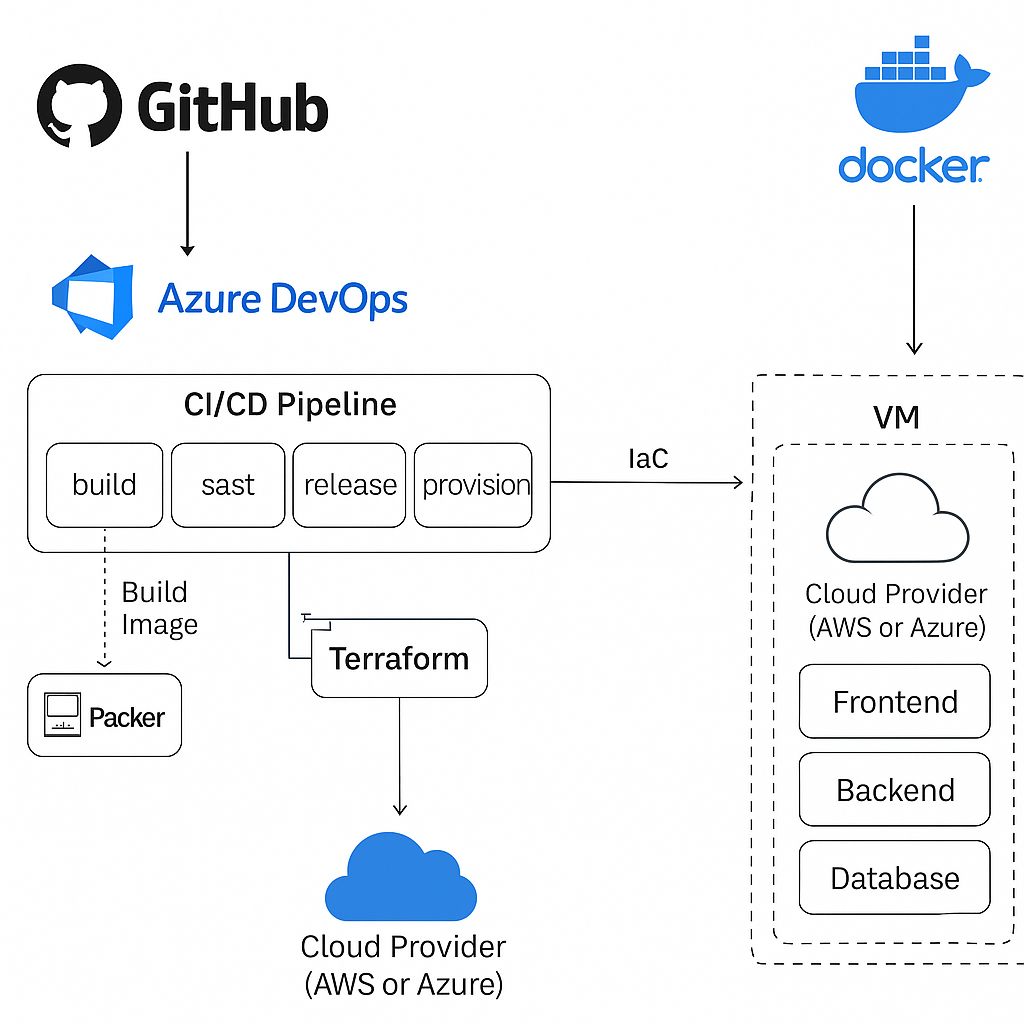


Ilustración 3 - Arquitectura del proyecto TFM Fuente: Elaboración propia

## Evaluación

La evaluación del sistema desarrollado tiene como objetivo verificar su aplicabilidad, usabilidad, robustez técnica y alineación con los principios del enfoque DevSecOps. La solución fue diseñada para ser modular, reproducible y adaptable a diferentes entornos, integrando tecnologías de infraestructura como código (Terraform), automatización de configuración (Ansible), creación de imágenes seguras (Packer), orquestación con contenedores (Docker), y flujos de CI/CD definidos por código en Azure DevOps.

La validación se llevó a cabo en escenarios realistas de despliegue, sobre infraestructuras provistas en AWS y Azure, con ejecución completa del pipeline desde la compilación hasta la verificación postdespliegue.(Accediendo a una url donde se despliega el aplicativo).

Para destacar las mejoras obtenidas con la implementación del pipeline CI/CD, se presenta una tabla comparativa que contrasta el enfoque manual tradicional de despliegue con el proceso automatizado desarrollado en este trabajo. Los criterios evaluados abarcan desde el tiempo de ejecución y la intervención humana hasta la seguridad y trazabilidad, permitiendo visualizar con claridad el valor agregado que aporta el modelo DevSecOps implementado.

|  |  |  |
| --- | --- | --- |
| **Criterio** | **Despliegue Manual** | **Pipeline DevSecOps Automatizado** |
| Tiempo de despliegue total | 2–3 horas | 10–15 minutos |
| Intervención humana | Alta | Nula |
| Reproducibilidad | Baja | Alta |
| Seguridad aplicada | Parcial (manual) | Integrada desde IaC |
| Validación post-despliegue | Manual | Automática |
| Trazabilidad | Limitada | Completa (logs, versiones) |

Tabla 6 - Comparación entre despliegue manual vs automatizado Fuente: Elaboración propia

### Usabilidad y aplicabilidad de la solución

El sistema demuestra una estructura clara y orientada a facilitar su adopción por parte de equipos técnicos con conocimientos básicos o intermedios en automatización. Los distintos componentes frontend, backend y base de datos están desacoplados y definidos como servicios Docker, lo que simplifica su entendimiento, despliegue y pruebas. La posibilidad de ejecutar el conjunto en docker-compose para validación local facilita aún más su uso como entorno de desarrollo o laboratorio.

El uso de GitHub como repositorio principal de código fuente y de configuraciones proporciona un entorno familiar y accesible, integrándose de forma nativa con Azure DevOps para ejecutar el pipeline CI/CD.

El sistema ha sido probado en entornos reales de AWS y Azure, utilizando imágenes generadas con Packer y aprovisionadas con Terraform. Esta capacidad de despliegue multicloud refleja su aplicabilidad en contextos organizativos con diferentes plataformas o estrategias híbridas.

### Pruebas unitarias, de integración y validación funcional

La evaluación técnica se apoyó en la ejecución automatizada de pruebas en cada fase del pipeline, cubriendo:

* **Pruebas unitarias del backend Java**:
  + Ejecutadas durante el stage build con mvn test.
  + Cobertura sobre endpoints REST y lógica de negocio básica.
  + Reportes generados como parte del pipeline y almacenados como artefactos.
* **Pruebas de integración entre componentes**:
  + Validación del flujo de datos desde el frontend hasta la base de datos.
  + Verificación de conexión, errores HTTP, y persistencia.
  + Ejecución en entorno local con docker-compose y en entorno remoto tras despliegue.
* **Pruebas funcionales del proceso de despliegue**:
  + Transferencia del .jar al servidor remoto.
  + Copia de seguridad de versiones previas.
  + Reinicio de la aplicación mediante systemctl o docker restart.
  + Verificación automática mediante un health check (curl /actuator/health).

Estas pruebas se integran en el pipeline como etapas independientes, lo que permite validar en cada ejecución si la aplicación compila, se comporta correctamente y puede ser desplegada sin intervención manual.

### Análisis de cumplimiento de objetivos

Para validar que el sistema cumple con lo planteado desde el diseño, se evaluó el grado de cumplimiento de cada uno de los objetivos específicos definidos en el Capítulo 3. La siguiente tabla resume esta correspondencia:

|  |  |  |
| --- | --- | --- |
| **Objetivo específico** | **Acción ejecutada** | **Evidencia obtenida** |
| Definir un flujo DevSecOps automatizado con Packer y Terraform | Diseño modular del pipeline en YAML | Pipeline funcional en Azure DevOps |
| Implementar infraestructura como código para entornos multicloud | Despliegue de VMs y redes en plataformas locales y extrapolables a AWS/Azure | Recursos provisionados correctamente y replicables |
| Integrar herramientas de escaneo y validación de seguridad | Análisis con Trivy, tfsec y SpotBugs | Reportes automáticos sin vulnerabilidades críticas |
| Aplicar políticas de hardening con Ansible | Playbooks específicos por rol: frontend, backend, base de datos | Configuraciones reforzadas con validación automatizada |
| Demostrar portabilidad y reproducibilidad del sistema | Repetición del despliegue en diferentes entornos de prueba | Consistencia de resultados entre ejecuciones, documentación completa |

Tabla 7 - Cumplimiento de objetivos específicos vs resultados Fuente: Elaboración propia

### Valor añadido de la solución desarrollada

Más allá del éxito técnico, el sistema aporta un valor significativo al integrar prácticas modernas de desarrollo y operaciones en una solución completa y práctica. Las herramientas utilizadas permiten reproducir el flujo en contextos educativos, institucionales o empresariales, sin barreras tecnológicas ni económicas.

Además, la integración de Packer y Terraform refuerza la consistencia del entorno desde su creación, y el uso de scripts en lugar de tareas predefinidas ofrece total transparencia y control. Esto permite que la solución pueda ser adoptada como referencia para proyectos que buscan implementar automatización segura desde cero, o como base para cursos de formación en DevOps, cloud computing y seguridad en pipelines.

Los resultados alcanzados no solo evidencian una mejora técnica en el proceso de despliegue, sino que también muestran el potencial de esta solución como modelo operativo replicable. Su diseño modular y orientado a herramientas open source lo convierte en una guía práctica para instituciones que buscan iniciar o reforzar su transición hacia modelos de automatización y DevSecOps, especialmente aquellas que aún operan con arquitecturas tradicionales y con equipos reducidos.

Además, su aplicabilidad se extiende a entornos educativos, donde puede ser utilizado como laboratorio de prácticas o base para proyectos de mejora continua.

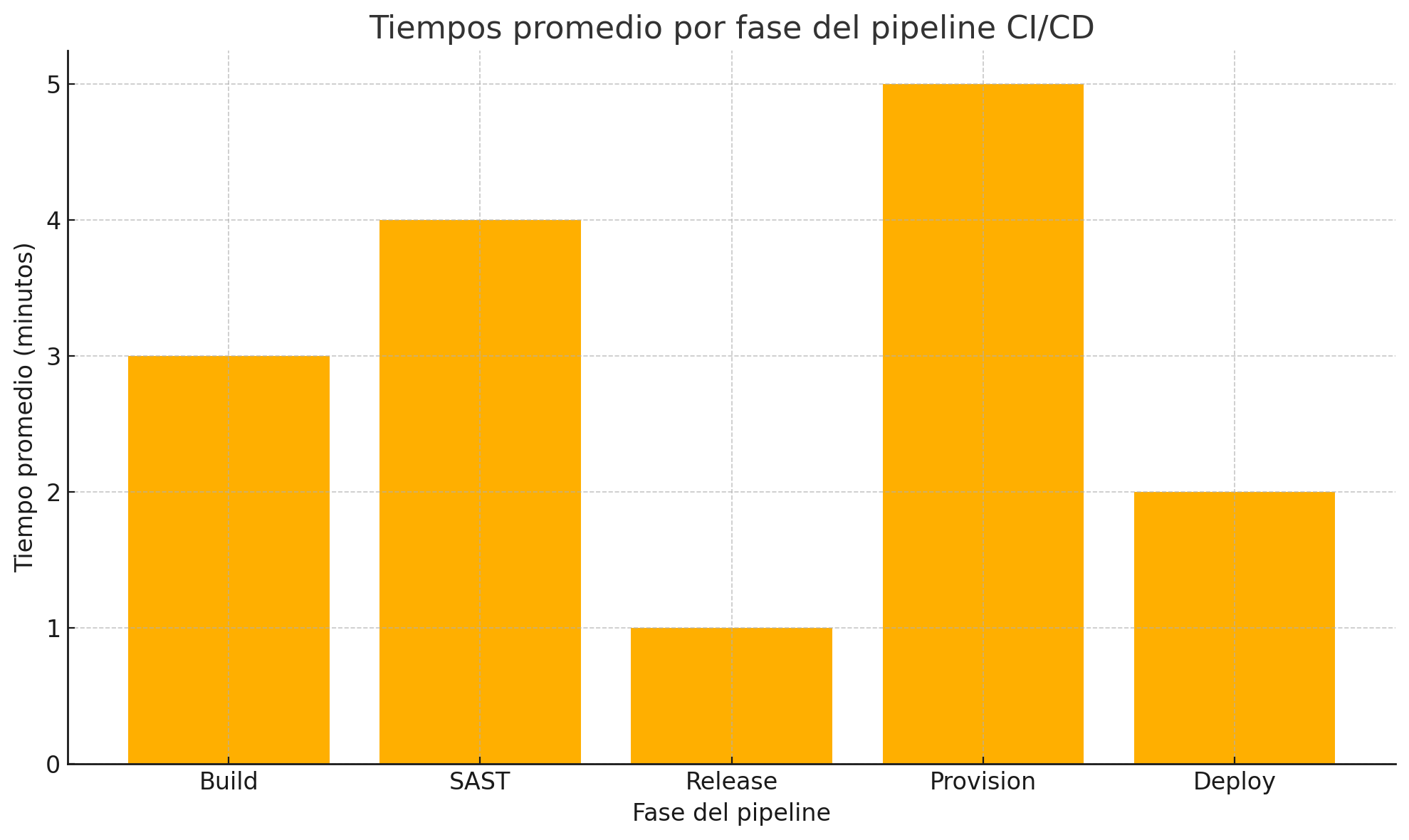


Ilustración 4 - Tiempos promedio por fase del pipeline Fuente: Elaboración propia

## Limitaciones del sistema desarrollado

El desarrollo del sistema propuesto en este Trabajo Fin de Máster se basó en principios sólidos de automatización, seguridad y reproducibilidad. A través de la integración de herramientas como Terraform, Ansible, Trivy y Azure DevOps, se logró implementar un pipeline funcional que permite desplegar de forma segura una arquitectura multicapa compuesta por frontend, backend y base de datos. No obstante, como parte de un ejercicio académico y aplicado, resulta indispensable analizar con objetividad las limitaciones que acompañaron a la construcción de esta solución.

Identificar las limitaciones no implica debilitar los logros alcanzados, sino más bien reconocer con honestidad las fronteras del alcance técnico y operativo, con miras a futuras mejoras y extensiones del sistema. En esta sección se exponen los factores que condicionaron la implementación, como la ausencia de ciertos componentes avanzados, decisiones de diseño orientadas a simplificar el entorno de pruebas, y restricciones inherentes al contexto de laboratorio utilizado.

Asimismo, se abordan aspectos relacionados con la escalabilidad, la seguridad avanzada, la observabilidad del sistema y la portabilidad hacia entornos de producción o arquitecturas cloud más complejas. También se reflexiona sobre los desafíos de adopción práctica en organizaciones reales, especialmente en aquellas con menor madurez tecnológica o recursos limitados para la formación de sus equipos.

La presentación estructurada de estas limitaciones permite no solo evaluar con mayor transparencia el trabajo realizado, sino también proyectar con claridad las oportunidades de mejora, evolución técnica y aplicabilidad futura del sistema desarrollado.

### Alcance limitado del entorno de despliegue

El pipeline fue diseñado para ejecutarse en un entorno de laboratorio controlado, con un número reducido de servidores virtuales y sin una orquestación de contenedores como Kubernetes. Si bien esto permitió mantener el enfoque pedagógico y técnico controlado, también implica que:

* No se evaluó la escalabilidad horizontal del sistema.
* No se incluyeron prácticas avanzadas como autoescalado, balanceo de carga o recuperación ante fallos en tiempo real.
* La solución, aunque reproducible, requiere adaptación para su despliegue en arquitecturas basadas en microservicios o clústeres de producción.

### Curva de aprendizaje y dependencia técnica

El diseño del pipeline y la implementación de las herramientas (Terraform, Ansible, Azure DevOps) demandan un conocimiento técnico intermedio o avanzado. Esto puede representar una barrera para su adopción inmediata en instituciones con equipos de TI no especializados o con restricciones de formación continua.

* La curva de aprendizaje de las herramientas IaC es considerable para personal no habituado a la automatización.
* La integración entre plataformas requiere una comprensión clara de variables, entornos y control de versiones.
* La documentación asociada, aunque disponible en el repositorio, puede ser ampliada con tutoriales interactivos o manuales más accesibles.

### Ausencia de herramientas de monitoreo centralizado

El alcance del TFM y del sistema no integró soluciones completas de observabilidad o monitoreo continuo como Prometheus, Grafana, o Zabbix. Tampoco se incluyeron herramientas de log centralizado como ELK Stack o Fluentd, aunque se dejaron espacios definidos para su incorporación futura, Sin embargo, esto no quiere decir que no se puedan establecer en sistemas reales.

* Esto implica que el análisis post-despliegue se basó en logs individuales y comandos puntuales.
* La supervisión del estado del sistema en tiempo real requeriría herramientas adicionales.

### Seguridad en etapas avanzadas

Si bien se aplicaron validaciones de seguridad con herramientas como tfsec, Trivy y SpotBugs, y se aplicaron configuraciones de hardening con Ansible, aún se pueden considerar mejoras como:

* Integración de análisis dinámico (DAST) posterior al despliegue.
* Validación de cumplimiento normativo mediante plataformas como OpenSCAP o Inspec.
* Gestión de secretos más robusta (uso de HashiCorp Vault u otras alternativas) en lugar de variables de entorno codificadas.

### Infraestructura sin alta disponibilidad (HA)

El sistema implementado no contempla configuraciones de alta disponibilidad (HA), replicación de base de datos o tolerancia a fallos. En un entorno de producción real, estos aspectos son esenciales para garantizar continuidad del servicio, especialmente en entornos críticos. Y estos deben ser abordados en base a las configuraciones específicas de cada sistema.

### Dependencia de servicios cloud externos

La solución fue pensada para ser extrapolada a entornos cloud como Azure y AWS. No obstante, esta portabilidad exige:

* Credenciales correctamente gestionadas.
* Configuración específica por proveedor.
* Consideraciones sobre costos, cuotas y permisos en cada cuenta.

Si bien el sistema está preparado técnicamente para este salto, su adopción final depende del contexto organizativo y la política de servicios en la nube de cada institución.

# Conclusiones y trabajo futuro

## Conclusiones

El presente Trabajo Fin de Máster ha tenido como finalidad diseñar, implementar y validar un flujo automatizado DevSecOps que integre herramientas de código abierto como Packer, Terraform, Ansible y gestionado desde GitHub y desplegado mediante un pipeline CI/CD modular en Azure DevOps, con el objetivo de gestionar infraestructura segura, portable y reproducible en entornos multicloud.

A partir del problema identificado la falta de soluciones automatizadas, seguras y replicables para entornos institucionales con madurez tecnológica variable se planteó una propuesta orientada a cerrar esa brecha, combinando tecnologías libres y buenas prácticas del ámbito DevSecOps. El resultado ha sido un sistema modular, técnicamente sólido y viable para su adopción en instituciones públicas, privadas y académicas.

Además de cumplir los objetivos específicos, el trabajo aporta una contribución práctica con impacto potencial en diversos entornos reales. La solución propuesta puede ser utilizada tanto como referencia educativa en programas de formación técnica, como punto de partida para estrategias de automatización en instituciones con limitaciones presupuestarias o sin acceso a herramientas propietarias.

Desde una perspectiva personal, este proyecto ha permitido consolidar conocimientos adquiridos durante el máster, enfrentando retos técnicos y metodológicos que reflejan situaciones reales en el campo profesional. Asimismo, refuerza la necesidad de fomentar espacios de formación práctica como los dojos tecnológicos que permitan a los equipos aprender, experimentar y aplicar metodologías modernas de forma colaborativa.

Por medio de esta propuesta aquí desarrollada considero que constituye una contribución práctica y replicable al campo de DevSecOps, aportando un modelo completo que integra infraestructura como código, contenerización y pipelines de CI/CD, y que cierra la brecha entre la teoría revisada en el estado del arte y su aplicación efectiva en el manejo de nubes híbridas.

## Líneas de trabajo futuro

El desarrollo de este Trabajo Fin de Máster ha permitido constatar que es viable automatizar de forma segura la gestión de infraestructura multicloud, cumpliendo con los principios del enfoque de DevSecOps. Esta aportación, más allá de su validez técnica, abre la puerta a una serie de aplicaciones prácticas y posibles mejoras que podrían enriquecer el campo del estudio y su adopción en distintos entornos organizativos.

En primer lugar, el sistema desarrollado podría servir como base formativa para entornos académicos o de capacitación técnica, especialmente en instituciones que buscan formar perfiles en automatización, seguridad e infraestructura como código. Su estructura modular, documentación clara y ejecución en entornos controlados lo convierten en un candidato ideal para ser utilizado en laboratorios, talleres o dojos tecnológicos orientados a la práctica.

Asimismo, el trabajo puede ser de utilidad para instituciones públicas o privadas que se encuentran en fases iniciales de transformación digital, especialmente aquellas que necesitan incorporar mecanismos de seguridad automatizada sin depender de tecnologías propietarias o infraestructuras costosas. La portabilidad y adaptabilidad del sistema permiten replicarlo en distintos escenarios y ajustarlo a las necesidades particulares de cada contexto.

Desde un enfoque más técnico, la propuesta desarrollada puede ser extendida para integrarse con herramientas adicionales de monitoreo, control de acceso, o despliegue sobre contenedores, permitiendo así una evolución natural hacia entornos más complejos como Kubernetes o arquitecturas serverless, sin perder la esencia de la automatización segura.

En términos generales, esta contribución se alinea con las demandas actuales del sector tecnológico, donde la seguridad desde el diseño, la infraestructura reproducible y la automatización transversal son pilares fundamentales. Por ello, el trabajo aquí desarrollado no debe verse como una solución cerrada, sino como un punto de partida sobre el que construir futuras iniciativas, tanto en el ámbito profesional como en el educativo, y que puede ser adaptado y escalado según las necesidades emergentes del ecosistema digital.

Referencias bibliográficas

1. Adewusi, A. (2023). The emergence and importance of DevSecOps: Integrating and reviewing security practices within the DevOps pipeline. [Repositorio académico o editorial si se conoce].
2. Alqahtani, H., Salah, K., Jayaraman, R., & Yaqoob, I. (2021). Secure cloud resource orchestration in multi-cloud using policy-based blockchain. Journal of Cloud Computing, 10(1), 1–18. https://doi.org/10.1186/s13677-021-00254-2
3. Capital One Tech Blog. (2021). Scaling security through infrastructure as code. https://medium.com/capital-one-tech
4. Chiari, M., De Pascalis, M., & Pradella, M. (2022). Static analysis of infrastructure as code: A survey. arXiv preprint. arXiv:2206.10344. https://doi.org/10.48550/arXiv.2206.10344
5. Morris, K. (2016). Infrastructure as code: Managing servers in the cloud. O’Reilly Media.
6. Myrbakken, H., & Colomo-Palacios, R. (2017). DevSecOps: A multivocal literature review. International Journal of Information Systems and Project Management, 5(3), 25–48. https://doi.org/10.12821/ijispm050302
7. zdoğan, E., Ceran, O., & Üstündağ, M. T. (2023). Systematic analysis of infrastructure as code technologies. GU Journal of Science, 10(4), 452–471.
8. Paul, A. L. (2024). Security challenges and solutions in multi-cloud environments. ResearchGate. [Añadir DOI o URL si se conoce]
9. Sandu, A. K. (2021). DevSecOps: Integrating security into the DevOps lifecycle for enhanced resilience. Technology and Management Review, 6(1). [Añadir página o URL si aplica]
10. Sharma, S., Sood, S. K., & Gupta, S. K. (2020). Secure cloud infrastructure: A DevSecOps approach. Journal of Cloud Computing, 9(1), 1–20. https://doi.org/10.1186/s13677-020-00171-5
11. Williams, L., & Fernández, M. A. (2019). Continuous security: DevSecOps and automated security verification. IEEE Software, 36(5), 88–92. https://doi.org/10.1109/MS.2018.2884925
12. Netflix Tech Blog. (2020). Security at Netflix: Open source tools. https://netflixtechblog.com
13. Repositorio de Código

Url del proyecto desarrollado.

<https://github.com/jorgearmijo882/DevOps_TFM.git>

1. Estructura del proyecto

El proyecto DevOps\_TFM implementa una solución práctica y completa enfocada en la automatización y seguridad mediante el enfoque DevSecOps, gestionando infraestructura como código para aplicaciones multicapa. El sistema está estructurado en componentes modulares que permiten su ejecución, facilitando el desarrollo y la validación inicial. Integra un backend construido con Java y Spring Boot, un frontend estático, y una base de datos PostgreSQL inicializada mediante scripts SQL. Además, el proyecto cuenta con herramientas avanzadas como Packer y Terraform para automatizar la creación de imágenes base y desplegar infraestructura consistente en diferentes plataformas como AWS, Azure o ambientes locales con VirtualBox. Todo esto es orquestado mediante pipelines de integración y entrega continua (CI/CD) definidos en Azure DevOps, asegurando la reproducibilidad, portabilidad y seguridad del proceso completo a continuación, se describe el contenido del proyecto:

* **Archivos Principales:**
* **README.md**: Contiene la descripción general, instrucciones básicas para el uso y configuración del proyecto.
* **docker-compose.yml**: Define la composición de contenedores Docker necesarios para levantar el proyecto localmente. (solución para despliegue en contenedores)
* **Backend (backend/):**

Este directorio aloja el código de la aplicación backend desarrollada en Java con Spring Boot.

* **Dockerfile**: Archivo que define la creación del contenedor Docker del backend.
* **pom.xml**: Archivo de gestión de dependencias y configuración del proyecto Maven.
* **src/main/java/com/example/inventory/**: Código fuente del backend.
  + **InventoryBackendApplication.java**: Clase principal de arranque de la aplicación Spring Boot.
  + **config/WebConfig.java**: Clase de configuración específica del backend.
  + **controller/ProductController.java**: Controlador REST para gestionar productos.
  + **model/Product.java**: Modelo que representa los productos en la aplicación.
  + **repository/ProductRepository.java**: Repositorio para operaciones CRUD de los productos.
* **src/main/resources/application.properties**: Archivo de configuración de Spring Boot (conexión a base de datos, servidor web, etc.).
* **Frontend (frontend/):**

Aquí se ubica el código del frontend estático del proyecto.

* **Dockerfile**: Define la creación del contenedor Docker del frontend utilizando probablemente Nginx.
* **index.html**: Página inicial del frontend de la aplicación.
* **Base de Datos (db/):**

Este directorio contiene scripts iniciales para la base de datos.

* **init.sql**: Script SQL para inicializar la estructura y datos iniciales de la base de datos PostgreSQL.
* **Infraestructura (infra/):**

Directorio destinado a la gestión automatizada de infraestructura con Packer y Terraform para diferentes proveedores.

* **packer/template.pkr.hcl**: Plantilla para la creación automatizada de imágenes base con Packer.
* **terraform/aws/main.tf**: Configuración Terraform para desplegar recursos en AWS.
* **terraform/azure/main.tf**: Configuración Terraform para desplegar recursos en Azure.
* **terraform/virtualbox/main.tf**: Configuración Terraform para desplegar y gestionar máquinas virtuales locales en VirtualBox.
* **Pipelines CI/CD (azure-pipelines/):**
* **pipeline.yml**: Definición del pipeline CI/CD para Azure DevOps que automatiza la compilación, pruebas y despliegue continuo del proyecto.

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Ilustración 5 - Estructura archivos Proyecto Fuente: Elaboración propia

1. Capturas de imagen del proyecto

Detalle del contenido del proyecto

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

En la carpeta de azure-pipelines se describe la creación de los pipelines para cada uno de los componentes del fullstak desarrollado.

En el backend se procede a utilizar la imagen de un contenedor versionada en el Docker-Hub.

Texto

El contenido generado por IA puede ser incorrecto.

En el bdd se procede a utilizar la imagen de un contenedor versionada en el Docker-Hub.

Texto

El contenido generado por IA puede ser incorrecto.

En el frontend se procede a utilizar la imagen de un contenedor versionada en el Docker-Hub.

Texto

El contenido generado por IA puede ser incorrecto.

En el Docker-compose.yml se procede con la automatización con Doker de forma local obteniendo las imágenes de Docker-Hub.

Texto

El contenido generado por IA puede ser incorrecto.

En el directorio de docker se procede con la declaración de los contenedores para uso de de forma local u cualquier proveedor en la nube.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

En el directorio de packer se procede con la declaración de los archivos para la creación de las maquinas backend, frontend, bdd en VirtualBox de forma local o en las nubes de Amazon con EC2.

Texto

El contenido generado por IA puede ser incorrecto.

En el directorio de terraform se procede con la declaración de los archivos para la creación de las maquinas backend, frontend, bdd en VirtualBox de forma local o en las nubes de Amazon con EC2.

Texto

El contenido generado por IA puede ser incorrecto.