UNIVERSIDAD TECNOLÓGICA ISRAEL

ESCUELA DE POSGRADOS "ESPOG"

MAESTRÍA EN SEGURIDAD INFORMÁTICA

Resolución: RPC-SO-02-No.053-2021

PROYECTO DE TITULACIÓN EN OPCIÓN AL GRADO DE MAGISTER

Título del proyecto:
Propuesta de un modelo de seguridad DevSecOps para detección y mitigación automatizada de vulnerabilidades en arquitecturas de microservicios On-Premise.
Línea de Investigación:
Ciencias de la ingeniería aplicada a la producción, sociedad y desarrollo sustentable
Campo amplio de conocimiento:
Tecnologías de la información y comunicación (TIC)
Autor/a:
Stalin Mauricio Mejía Montenegro
Tutores:
PhD. Renato Toasa PhD Urdaneta Herrera Maryory

Quito – Ecuador

2025

APROBACIÓN DEL TUTOR

Yo, **Toasa Guachi Renato Mauricio** con C.I: **1804724167** en mi calidad de Tutor del proyecto de investigación titulado: PROPUESTA DE UN MODELO DE SEGURIDAD DEVSECOPS PARA DETECCIÓN Y MITIGACIÓN AUTOMATIZADA DE VULNERABILIDADES EN ARQUITECTURAS DE MICROSERVICIOS ON-PREMISE.

Elaborado por: **Mejía Montenegro Stalin Mauricio**, de C.I: 1725667065, estudiante de la Maestría: Seguridad Informática, de la **UNIVERSIDAD TECNOLÓGICA ISRAEL (UISRAEL)**, como parte de los requisitos sustanciales con fines de obtener el Título de Magister, me permito declarar que luego de haber orientado, analizado y revisado el trabajo de titulación, lo apruebo en todas sus partes.

Quito D.M., septiembre de 2025

Firmado electrónicamente por:
RENATO MAURICIO TOASA GUACHI
Validar únicamente con FirmaEC

_____

PhD. Toasa Guachi Renato Mauricio

Firma

APROBACIÓN DEL TUTOR

Yo, **Maryory Urdaneta Herrera** con C.I: **1759316126** en mi calidad de Tutor del proyecto de investigación titulado: PROPUESTA DE UN MODELO DE SEGURIDAD DEVSECOPS PARA DETECCIÓN Y MITIGACIÓN AUTOMATIZADA DE VULNERABILIDADES EN ARQUITECTURAS DE MICROSERVICIOS ON-PREMISE.

Elaborado por: **Mejía Montenegro Stalin Mauricio**, de C.I: 1725667065, estudiante de la Maestría: Seguridad Informática, de la **UNIVERSIDAD TECNOLÓGICA ISRAEL (UISRAEL)**, como parte de los requisitos sustanciales con fines de obtener el Título de Magister, me permito declarar que luego de haber orientado, analizado y revisado el trabajo de titulación, lo apruebo en todas sus partes.

Quito D.M., septiembre de 2025

MARYORY URDANETA HERRERA

Firmado digitalmente por MARYORY URDANETA HERRERA
Fecha: 2025.09.01 12:37:33 -05'00'

_____

PhD. Urdaneta Herrera Maryory

Firma

DECLARACIÓN DE AUTORIZACIÓN POR PARTE DEL ESTUDIANTE

Yo, Stalin Mauricio Mejía Montenegro con C.I: 1725667065, autor/a del proyecto de titulación denominado: PROPUESTA DE UN MODELO DE SEGURIDAD DEVSECOPS PARA DETECCIÓN Y MITIGACIÓN AUTOMATIZADA DE VULNERABILIDADES EN ARQUITECTURAS DE MICROSERVICIOS ON-PREMISE. Previo a la obtención del título de Magister en Seguridad Informática.

1. Declaro tener pleno conocimiento de la obligación que tienen las instituciones de educación superior, de conformidad con el Artículo 144 de la Ley Orgánica de Educación Superior, de entregar el respectivo trabajo de titulación para que sea integrado al Sistema Nacional de Información de la Educación Superior del Ecuador para su difusión pública respetando los derechos de autor.
2. Manifiesto mi voluntad de ceder a la Universidad Tecnológica Israel los derechos patrimoniales consagrados en la Ley de Propiedad Intelectual del Ecuador, artículos 4, 5 y 6, en calidad de autor@ del trabajo de titulación, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente. En concordancia suscribo este documento en el momento que hago entrega del trabajo final en formato impreso y digital como parte del acervo bibliográfico de la Universidad Tecnológica Israel.
3. Autorizo a la SENESCYT a tener una copia del referido trabajo de titulación, con el propósito de generar un repositorio que democratice la información, respetando las políticas de prosperidad intelectual vigentes.

Quito D.M., septiembre de 2025

Firmado electrónicamente por:
STALIN MAURICIO
MEJIA MONTENEGRO
Validar únicamente con FirmaEC

_____

Firma

Tabla de contenidos

Índice de tablas

Índice de figuras

INFORMACIÓN GENERAL

Contextualización del tema

En los últimos años, la forma en que los equipos desarrollan software ha cambiado a una velocidad acelerada. Los patrones de microservicios y los hábitos de DevOps han sustituido las antiguas rutinas en cascada, favoreciendo en su lugar canales flexibles y automatizados. Con la integración continua y la entrega continua funcionando en segundo plano, el código puede empaquetarse y enviarse a los clientes muchas veces al día.

Sin embargo, ese ritmo plantea un problema complejo: ¿cómo mantener la seguridad cuando los límites entre las etapas del ciclo de vida son cada vez más difusos?. En entornos locales distribuidos, cada microservicio arrastra consigo sus propias dependencias. Si se pasa por alto un solo fallo, los problemas pueden propagarse por los contenedores, colapsar los servicios y dañar toda la aplicación.

Por lo tanto, si bien el nuevo ritmo impulsa innegablemente la productividad y eficiencia, también reescribe las reglas del ciclo de vida del desarrollo de software (SDLC). El desarrollo, las pruebas y el lanzamiento ahora se desarrollan en paralelo; un error o una configuración incorrecta en un carril puede pasar a producción sin que nadie se dé cuenta de que se ha caído algún servicio.

Cuando las empresas adoptaron arquitecturas de microservicios, adquirieron la capacidad de moverse con rapidez, escalar bajo demanda y enviar actualizaciones a un ritmo acelerado. Lo que a menudo pasa desapercibido, al menos al principio, es la complejidad que conlleva, especialmente en el ámbito de la seguridad. Las tácticas que mantenían una aplicación clásica todo en uno razonablemente segura empiezan a desmoronarse una vez que el código base se fragmenta en docenas o cientos de pequeños servicios. Es menos una cuestión de descuido que un efecto secundario predecible del salto arquitectónico que hemos dado.

En los centros de datos locales, donde los sistemas están naturalmente más fragmentados, los riesgos son aún mayores. Si se filtra una vulnerabilidad, el radio de acción puede crecer rápidamente. La exposición prolongada de servicios críticos e interconectados no solo amenaza el tiempo de actividad, sino que puede agotar los presupuestos debido al trabajo de recuperación, la pérdida de ingresos y, lo que es peor, las multas regulatorias por incumplimiento. El daño intangible es más difícil de cuantificar: los socios dudan, los clientes buscan otras alternativas y la reputación ganada por las empresas se desmoronan. Dicho de otro

modo, integrar la seguridad en el trabajo diario no solo es una decisión tecnológica, si no una exigencia clave para la continuidad operacional en las empresas.

Simplemente duplicar las medidas de seguridad no será suficiente. Se tiene que replantear toda una estrategia segura para la mejora en la gestión de vulnerabilidades. Las aplicaciones desarrolladas con arquitecturas en microservicios utilizando contenedores, escalan horizontalmente o pueden desaparecen en minutos, a veces sin intervención humana, cada servicio es una puerta de entrada que un atacante podría manipular. La seguridad debe ser igual de ágil. DevOps sentó las bases, demostrando que la automatización, sumada a la integración y entrega continua, puede reducir los ciclos de lanzamiento de meses a horas. Sin embargo, esa misma velocidad puede introducir puntos débiles si las barreras de seguridad se quedan atrás. Intercambiar seguridad por velocidad es una falsa economía; se pagará caro, de una forma u otra.

Aparece DevSecOps, la idea principal es que la seguridad debe estar presente en cada fase, no como una capa final. Comienza desde el diseño y se mantiene activo durante la compilación, las pruebas, la implementación y la ejecución. En este punto, integrar la seguridad en el pipeline ya no es una mejora atractiva; es una apuesta segura. Los procesos ágiles dominan, y cualquier fallo detectado tarde costará más dinero, más tiempo y más buena voluntad. DevSecOps, entonces, no es el freno; es el sistema de suspensión que permite tomar las curvas rápidamente sin derrapar y mejorar la seguridad en cada fase del ciclo de desarrollo de software.

El aspecto cultural importa tanto como las herramientas. Desarrolladores, operadores y profesionales de seguridad necesitan reunirse desde el principio y con frecuencia, derribando los silos que solían separarlos. En configuraciones de microservicios locales, esa estrecha colaboración permite predecir puntos débiles, integrar comprobaciones automatizadas y reaccionar en tiempo real sin retrasar la entrega. En definitiva, DevSecOps es menos un marco y más una mentalidad que alinea la protección con la cadencia implacable del trabajo del software moderno.

Problema de investigación

Operar con un modelo de microservicios sin duda aumenta la eficiencia y la escalabilidad, pero también convierte la seguridad en un problema más complejo. Cada microservicio es una puerta de entrada que los atacantes pueden desestabilizar, y más aún en los centros de datos On-Premise, donde el hardware es más limitado que otros entornos como la nube pública, al no contar con estrategias de seguridad en la automatización e integración de arquitectura de microservicios, los problemas de seguridad aumentan obligando a las empresas a rediseñar la gestión de seguridad.

DevSecOps propone resolver este problema integrando seguridad, gestión, análisis y la aplicación de políticas en cada fase del ciclo de vida del software. Sin embargo, las empresas locales aún tienen dificultades para implementarlo. La escasa automatización, las cadenas de herramientas incompatibles y las persistentes limitaciones de la infraestructura impiden su adopción total.

Como resultado, muchas empresas se aferran a la vieja estrategia que servía a los sistemas monolitos, y esa estrategia simplemente no puede seguir el ritmo y el detalle de los microservicios, dejando grandes brechas por las que un atacante puede pasar.

¿Cómo diseñar un modelo DevSecOps que permita detectar y mitigar de forma automatizada las vulnerabilidades en arquitecturas de microservicios en entornos On-Premise, manteniendo la eficiencia y seguridad durante el ciclo de desarrollo de software?

Objetivo general

Proponer un modelo de seguridad DevSecOps que permita la detección y mitigación automatizada de vulnerabilidades en entornos On-Premise con arquitecturas de microservicios, integrándolo eficientemente en el ciclo de vida del desarrollo de software.

Objetivos específicos

- Analizar las vulnerabilidades más comunes asociadas a arquitecturas de microservicios en entornos On-Premise.
- Proponer lineamientos y buenas prácticas para la adopción del enfoque DevSecOps en entornos On-Premise.
- Desarrollar un modelo de Seguridad utilizando metodología DevSecOp para la detección y mitigación de vulnerabilidades en el proceso de integración y despliegue continuo.
- Validar el modelo propuesto mediante especialistas en un entorno de simulación On-Premise.

Vinculación con la sociedad y beneficiarios directos:

El modelo propuesto de seguridad DevSecOps se vincula directamente con un entorno laboral: se pretende implementar el plan modelo de seguridad en la empresa Aliservis S.A. Los primeros beneficiarios serán los equipos de TI: desarrolladores, ingenieros de operaciones y analistas de seguridad. Ayudarán a verificar y validar el modelo propuesto y, al mismo tiempo, mejorarán sus habilidades mediante procesos prácticos, guías y capacitaciones sobre cómo integrar la seguridad en cada fase del proceso de desarrollo.

Por otra parte, se realizó una difusión mediante un seminario web en el Instituto Superior Tecnológico Cordillera. Durante la sesión, se analizó los aspectos prácticos de la seguridad del ciclo de vida, se analizó cómo automatizar las comprobaciones, fases en ciclo de desarrollo de software y se reveló la importancia de DevSecOps en las empresas que aún utilizan su propia infraestructura. Estudiantes y profesores conocieron las tendencias actuales en ciberdefensa y adquirieron una mentalidad preventiva que podrán aplicar mucho antes de incorporarse al mercado laboral. Para la revisión del seminario web se adjunta link de conferencia. https://youtu.be/X5gnjNUAWkM

El plan pretende llegar a más de una empresa que utiliza actualmente arquitecturas microservicios en entornos On-Premise. Muchas empresas aún dependen de equipos locales; la misma estrategia permitirá orientarles a los beneficios propuestos en el modelo DevSecOps. Integrando componentes reutilizables (scripts de automatización, archivos de integración listos para usar, instrucciones claras y notas de refuerzo de seguridad basadas en herramientas de código abierto) para que cualquier equipo que gestione seguridad fuera de la nube pública pueda adoptarlos sin complicaciones.

En concreto, el modelo debería reforzar la estrategia de seguridad de Aliservis S.A, reducir los problemas operativos y optimizar las entregas y despliegues en aplicaciones con microservicios. Al impulsar a los equipos de desarrollo, operaciones y seguridad a avanzar en la misma dirección, también se fomenta una cultura DevSecOps duradera de codificación segura. Con el tiempo, esto podría tener repercusiones: menos errores en el software publicado, mayor confianza del cliente y un panorama empresarial mejor preparado para la próxima ola de ciberamenazas.

CAPÍTULO I: DESCRIPCIÓN DEL PROYECTO

1.1. Contextualización general del estado del arte

La transición de monolitos estrechamente acoplados a arquitecturas de microservicios ha obligado a los equipos de tecnología a integrar la seguridad en las primeras fases del ciclo de desarrollo de un sistema, en lugar de incorporarla en fases finales. Esta idea se refleja en la tesis de licenciatura de Quichimbo Guamán y Romero González (2024), donde implementaron una rutina DevSecOps en la aplicación VenceYa. Al integrar escaneos y comprobaciones automatizadas en cada confirmación, detectaron puntos débiles de forma temprana y redujeron considerablemente el riesgo general.

Un tema similar se refleja en un proyecto de maestría de 2025 en la Universidad Técnica del Norte. Dirigido al software bancario local, el estudio identificó a SonarQube, GitHub Actions, Docker y Trivy como los recursos de código abierto más fiables que detectan problemas antes de que el código entre en producción. Los autores demostraron que priorizar la seguridad no solo refuerza las defensas, sino que también reduce drásticamente los costos de las correcciones posteriores.

El trabajo de Vergara en la Universidad Técnica de Manabí (2023) llega a la misma conclusión desde otra perspectiva. Su propuesta se centró en automatizar el ciclo de integración y entrega continua, señalando la frecuencia con la que las brechas permanecen ocultas hasta la publicación de una versión. La conclusión: permitir que las máquinas detecten fallas mucho antes que los clientes.

Taibi y Lenarduzzi (2020) indagan que los "bad smells" en los microservicios que surgen una vez que un proyecto se divide en docenas de servicios independientes. Dado que cada microservicio opera con su propia línea de tiempo y expone su propia API, la superficie de ataque se expande y las defensas perimetrales se descomponen. Abogan por controles que profundicen en cada componente individual y puedan reproducirse automáticamente para el análisis de vulnerabilidades.

Tello Medina (2020) presenta un caso práctico para el análisis estático en el pipeline, señalando que la detección temprana evita que las fallas de seguridad se filtren en las compilaciones de producción. En configuraciones locales, las comprobaciones automatizadas son la solución ideal porque los equipos pequeños no pueden permitirse revisar cada fusión manualmente.

Schlattmann y Dolog (2021) profundizan en la idea "Seguridad como Código". Las políticas, los análisis y los scripts de cumplimiento conviven con el código de la aplicación, versionados y probados en las mismas solicitudes de extracción. Tratar los controles de seguridad automatizada, versionada y probada elimina los cuellos de botella manuales que antes paralizaban las versiones.

Para Fernández (2020), los aspectos económicos son innegables. Corregir una falla después de la puesta en marcha cuesta mucho más que detectarla en una prueba unitaria, y el impacto en la reputación puede colapsar ambos. Desde su perspectiva de gestión de riesgos, una postura de cambio "shift-left" se amortiza sola previniendo costos operativos y brechas de seguridad.

Una amplia revisión de Shahin et al. (2021) subraya la importancia de CI/CD. Automatizar integraciones, pruebas y versiones acorta los ciclos de retroalimentación, aumenta la fiabilidad y minimiza el error humano. Esas mismas automatizaciones son puntos perfectos para introducir tareas de DevSecOps sin ralentizar el proceso de seguridad.

Cusco Mejía (2022) se centra en la capa de infraestructura. Utilizando Docker, Kubernetes y scripts de infraestructura como código en hardware local, demuestra que los equipos pueden moverse a la velocidad de la nube sin dejar de alojar todo internamente. Los entornos con control de versiones controlan las desviaciones y simplifican las reversiones.

Finalmente, González y Quichimbo (2024) vuelven al mensaje principal: cuando las pruebas de seguridad automatizadas abarcan cada fase, desde el diseño hasta la implementación, las vulnerabilidades rara vez se detectan y los mayores riesgos se reducen a un tamaño manejable.

En conjunto, estos estudios presentan un panorama claro: la seguridad no puede esperar al último sprint. Automatizar las comprobaciones, integrar políticas como código y tratar cada microservicio como una fortaleza propia convierte a DevSecOps de una palabra de moda en un procedimiento operativo estándar, especialmente para las organizaciones que aún ejecutan sus servidores localmente.

1.2. Proceso investigativo metodológico

La investigación cualitativa es el procedimiento metodológico que utiliza palabras, textos, discursos, dibujos, gráficos e imágenes para construir un conocimiento de la realidad social, en un proceso de conquista-construcción-comprobación teórica desde una perspectiva holística, pues se trata de comprender el conjunto de cualidades interrelacionadas que caracterizan a un determinado fenómeno. La perspectiva cualitativa de la investigación intenta acercarse a la realidad social a partir de la utilización de datos no cuantitativos (Álvarez y Jurgenson, 2019).

La investigación sigue una línea principalmente cualitativa con un análisis no probabilístico, complementada con perspectivas selectivas y enmarcada como investigación aplicada. Su propósito es poner a prueba y validar un esquema DevSecOps en sistemas locales On-Premise mediante la observación directa, recopilación y análisis de datos relacionados con la detección, mitigación y gestión de vulnerabilidades. Junto con las métricas, los manuales pertinentes, los estándares de cumplimiento y las directrices del sector, orientarán la construcción del marco, lo que permitirá respaldar el diseño del modelo y la interpretación cualitativa de dónde el modelo funciona bien y dónde necesita ajustes.

1.2.1 Tipo de investigación

El tipo de investigación aplicada es exploratorio, descriptivo y aplicado. Es exploratorio porque examina el marco DevSecOps en etapas de desarrollo (CI – CD), dentro de las configuraciones de microservicios locales On-Premise. Es descriptivo porque se mapeará todo el proceso de desarrollo en los diferentes entornos: herramientas, transferencias y las vulnerabilidades de seguridad que tienden a aparecer. Por último, se trata de una investigación aplicada: el objetivo final es un plan de trabajo que automatice la detección y reparación de vulnerabilidades, ofreciendo ganancias concretas en protección, tiempo de respuesta y calidad general del código.

1.2.2 Población y muestra

La población de referencia comprende entornos técnicos locales que albergan arquitecturas de microservicios y dependen de pipelines de CI/CD con controles de seguridad integrados. La unidad de análisis es cada pipeline ejecutado junto con la información que genera: registros de compilación e implementación, registros de trazabilidad y auditoría, listas de vulnerabilidades, registros de infraestructura, paneles de monitoreo y metadatos de configuración. Las notas recopiladas durante iteraciones sucesivas permiten una evaluación empírica y repetible de la efectividad del marco DevSecOps propuesto.

Los datos se recopilan mediante la monitorización directa de las tareas del pipeline y el archivo sistemático de sus resultados, lo que permite comprobaciones paralelas a lo largo del tiempo. Dado que un control estricto de las variables es esencial, el estudio emplea una muestra intencional y no probabilística: un entorno de laboratorio que refleja con precisión las operaciones diarias locales.

Además, el estudio se centra en configuraciones locales de desarrollo y despliegue, ejecutadas por organizaciones que poseen y operan sus propios servidores. Para las pruebas,

utilizaremos una muestra intencional y no aleatoria: un entorno de pruebas que refleja una propuesta de implementación local con el modelo DevSecOps utilizando microservicios en el proceso de integración continua y despliegue continuo.

1.2.3 Métodos, técnicas e instrumentos

Para llevar a cabo el estudio, se empleará un diseño experimental. Al aplicar el plan de seguridad DevSecOps en un entorno de laboratorio que refleja las operaciones locales diarias, podemos observar el comportamiento de la configuración al detectar patrones de ataque típicos. Ejecutar la prueba en condiciones fijas y repetibles nos permite obtener cifras concretas sobre las tasas de alerta temprana, el tiempo medio de respuesta y la reducción de errores de codificación que pasan en cada sprint.

En cuanto a la recopilación de datos, se utilizará la observación directa y el registro que se obtiene de la ejecución de pipelines en el entorno CI/CD. Se registrará cada evento relevante antes y después de la puesta en marcha del marco, creando un entorno que nos permitirá extraer información valiosa. Los indicadores clave incluyen el número de fallos detectados automáticamente, el tiempo medio de parcheo, el volumen de alertas activadas y cualquier problema en la cadena de CI/CD. La comparación de estos indicadores mostrará con precisión dónde y cómo el nuevo proceso avanza.

Como complemento, se aplicará una revisión documental que fundamentará el marco del modelo DevSecOps. Analizaremos marco normativo DevSecOps Maturity Model, las recomendaciones de OWASP para las 10 principales aplicaciones web y API, y artículos recientes revisados por pares sobre DevSecOps, microservicios, CI/CD e investigación de vulnerabilidades. Integrar estas referencias en el diseño alinea nuestro prototipo con las normas actuales y las mejores prácticas avaladas globalmente, lo que refuerza tanto su solidez técnica como su potencial de adopción en el mundo real.

1.2.4 Instrumentos de recolección de datos

La observación es un método de recolección de datos utilizado en investigaciones sociales, psicológicas, antropológicas y en muchos otros campos. Se trata de una técnica que permite a los investigadores observar comportamientos, actitudes y eventos en un entorno natural o controlado. La observación es una técnica valiosa debido a su capacidad para proporcionar información objetiva y detallada sobre un sujeto o situación (Bastidas, 2019).

Para obtener pruebas sólidas de la efectividad del modelo, se utiliza el método de observación directa de varias herramientas de seguimiento que registran lo que sucede en cada etapa del ciclo DevSecOps.

Primero están los registros (bitácoras) de ejecución que Jenkins genera cada vez que se activa en cada pipeline de CI/CD. Estos registros describen los pasos de compilación, pruebas unitarias, análisis de seguridad e implementación para cada servicio, en tiempo real, lo que nos permite detectar fallos, medir tiempos de respuestas y evaluar la eficiencia diaria.

Además, los resultados de los reportes generados muestran información para el análisis de: SonarQube para comprobaciones estáticas, OWASP ZAP para sondeo en vivo y OWASP Dependency-Check para bibliotecas de terceros. Sus hallazgos se contabilizarán y ordenarán para que podamos ver exactamente qué fallos se esconden en el código o las dependencias, y cómo cambian esas cifras una vez que se activan las barreras de seguridad automatizadas.

En conjunto, los registros y los resúmenes de análisis ofrecen una sólida base cuantitativa. Al alinear instantáneas de antes y después, podemos medir el impacto del marco, confirmar su viabilidad y demostrar si realmente fortalece el proceso de desarrollo local.

1.3. Análisis de resultados

Los resultados obtenidos durante la observación directa demuestran, que el marco DevSecOps realmente refuerza la seguridad en las instalaciones locales On-Premise. Utilizando los instrumentos de recolección descritos anteriormente, validamos todas las fases de CI/CD en una pila de microservicios que se ejecutan dentro de un entorno de pruebas. Este ejercicio práctico nos permite observar el marco en acción y, aún más importante, cuantificar su influencia en tres áreas: la rapidez con la que se detectan las fallas, la cantidad de barreras de seguridad que se ejecutan sin intervención humana y la rapidez con la que el equipo técnico puede reaccionar cuando algo parece extraño.

La monitorización continua y la captura detallada de registros generaron un conjunto de datos sólido que confirma la capacidad del modelo para:

- Identificar debilidades en cada fase: código fuente mediante SAST, paquetes y contenedores de terceros mediante SCA, endpoints activos mediante DAST, todo ello sin necesidad de intervención manual.
- Cuantificar y clasificar los riesgos según su severidad (Crítico, Alto, Medio, Bajo).
- Imponer políticas de seguridad de manera automatizada a través de un Quality Gate, deteniendo el despliegue de código no seguro.

La figura 1 corresponde a la bitácora de ejecución del pipeline CI/CD generada por Jenkins, uno de los principales instrumentos de recolección de datos. Se observa la secuencia de etapas automatizadas, desde la compilación hasta el análisis de seguridad Sonar Analyst, Trivy Scan. El dato más relevante es que la ejecución falló y se detuvo automáticamente después del análisis de SonarQube, impidiendo que las etapas posteriores como Docker Build y Deploy se ejecutaran. Esto demuestra empíricamente que el modelo es funcional y cumple su objetivo de bloquear la entrega de software que no cumple con los criterios de seguridad establecidos.

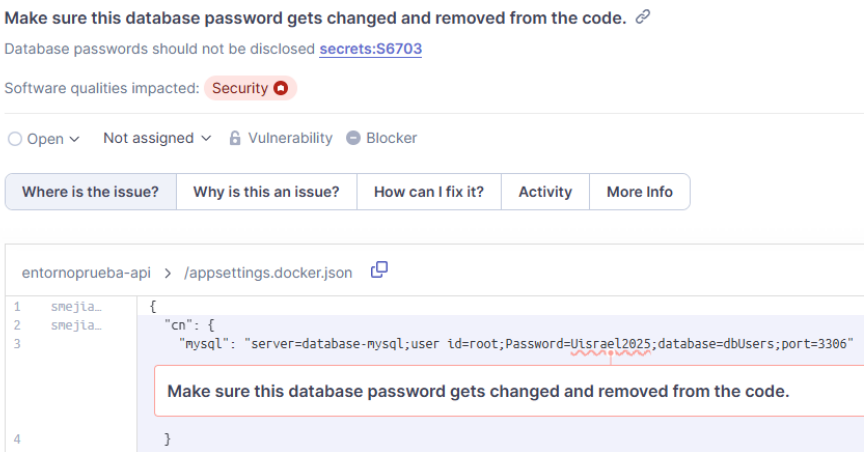
Figura 1.

Análisis del Pipeline CI/CD con Jenkins entorno de pruebas controlado

La Figura 2 muestra el resultado generado por SonarQube, el motor de análisis estático (SAST) utilizado en el estudio. En esta ejecución, la herramienta genera una alerta de "Bloqueador" (de gravedad alta) tras detectar una contraseña de base de datos codificada de forma rígida en el archivo de configuración appsettings.docker.json. El registro y la clasificación de esta falla confirman la capacidad del framework para detectar defectos críticos durante las primeras etapas del pipeline. El resultado se alinea con el diseño de la investigación, que priorizó los análisis automatizados dentro del flujo de CI/CD para detectar problemas lo antes posible.

Figura 2.

Análisis Estático de Código (SAST) con SonarQube



La figura 3 muestra una evidencia directa que conecta los dos puntos anteriores. El mensaje "Quality gate is ERROR'" confirma que el resultado del análisis de SonarQube activó la política de seguridad. Inmediatamente después, el log indica que la etapa Docker Build fue omitida debido a esta falla. Este registro sistemático es crucial, ya que documenta el comportamiento del sistema y demuestra la integridad del proceso CI/CD, donde la detección de una vulnerabilidad tiene una consecuencia automática y trazable.

Figura 3.

Evidencia de Falla del Quality Gate en el Log

```
SonarQube task 'AZiGoddQqjV6z3xvz-e2' completed. Quality gate is 'ERROR'
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Docker Build)
Stage "Docker Build" skipped due to earlier failure(s)
```

En la figura 4 muestra reportes generados por Trivy, la herramienta que cumple la función de Análisis de Composición de Software (SCA) para el contenedor. El primer informe resume que se encontraron 8 vulnerabilidades en la imagen base de Debian. El segundo informe detalla estas vulnerabilidades, clasificándolas por severidad e identificando los CVE (Common Vulnerabilities and Exposures) correspondientes, como una vulnerabilidad CRÍTICA (CVE-2023-45853) en la librería zlib1g y una ALTA (CVE-2025-4802) en libc-bin. Estos datos cuantificables validan la capacidad del modelo para analizar no solo el código de la aplicación, sino también las dependencias del entorno de ejecución.

Figura 4.

Análisis de Dependencias y Contenedores (SCA) con Trivy

Report Summary

Target	Type	Vulnerabilities	Secrets
stalin9116/entornoprueba-api:latest (debian 12.11)	debian	8	-
app/WebApplication1.deps.json	dotnet-core	0	-
usr/share/dotnet/shared/Microsoft.AspNetCore.App/8.0.18/Microsoft.AspNetCore.App.deps.json	dotnet-core	0	-
usr/share/dotnet/shared/Microsoft.NETCore.App/8.0.18/Microsoft.NETCore.App.deps.json	dotnet-core	0	-

stalin9116/entornoprueba-api:latest (debian 12.11)

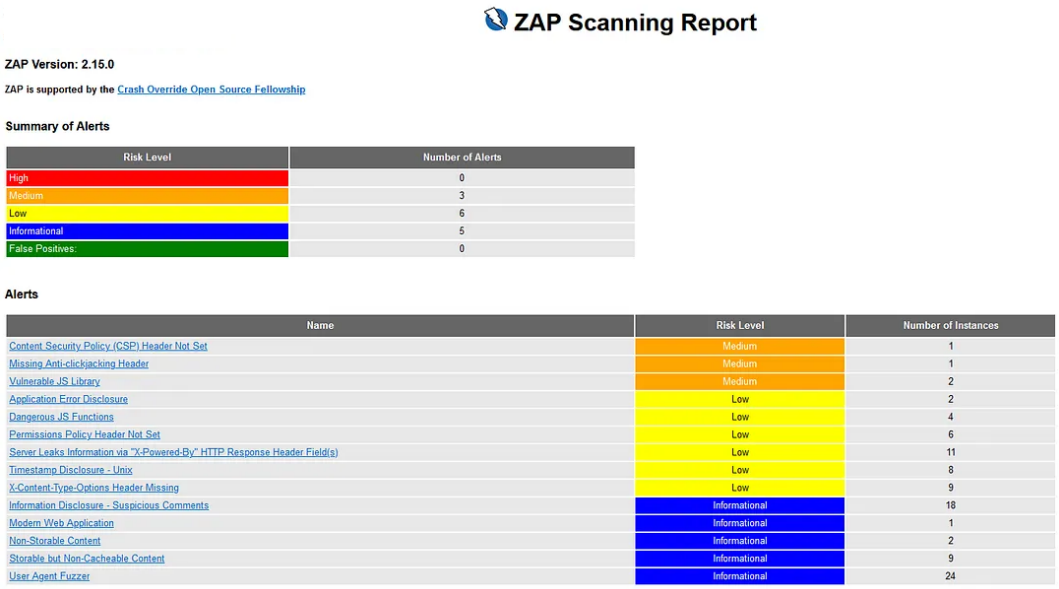
Total: 8 (HIGH: 7, CRITICAL: 1)

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
libc-bin	CVE-2025-4802	HIGH	affected	2.36-9+deb12u10		glibc: static setuid binary dlopen may incorrectly search LD_LIBRARY_PATH https://avd.aquasec.com/nvd/cve-2025-4802
libc6						
libpam-modules	CVE-2025-6020			1.5.2-6+deb12u1		linux-pam: Linux-pam directory Traversal https://avd.aquasec.com/nvd/cve-2025-6020
libpam-modules-bin						
libpam-runtime						
libpam0g						
perl-base	CVE-2023-31484			5.36.0-7+deb12u2		perl: CPAN.pm does not verify TLS certificates when downloading distributions over HTTPS... https://avd.aquasec.com/nvd/cve-2023-31484
zlib1g	CVE-2023-45853	CRITICAL	will_not_fix	1:1.2.13.dfsg-1		zlib: integer overflow and resultant heap-based buffer overflow in zipOpenNewFileInZip4_6 https://avd.aquasec.com/nvd/cve-2023-45853

La Figura 5 muestra el resultado de OWASP ZAP, la utilidad de pruebas dinámicas (DAST) utilizada en este estudio. El análisis cataloga las debilidades detectadas durante la interacción en vivo con el servicio. Si bien no se observaron advertencias de gravedad alta, sí se identificaron tres problemas de nivel medio como una "Biblioteca JavaScript vulnerable" y la ausencia de un encabezado "Content-Security-Policy", junto con seis hallazgos con una calificación baja. Estos resultados destacan el amplio alcance del marco de trabajo; al exponer fallos de configuración y tiempo de ejecución que las herramientas estáticas pasan por alto, se obtiene una visión completa de la seguridad del sistema.

Figura 5.

Análisis Dinámico de Aplicación (DAST) con OWASP ZAP



ZAP Scanning Report

ZAP Version: 2.15.0

ZAP is supported by the Crash Override Open Source Fellowship

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	3
Low	6
Informational	5
False Positives:	0

Alerts

Name	Risk Level	Number of Instances
Content Security Policy (CSP) Header Not Set	Medium	1
Missing Anti-clickjacking Header	Medium	1
Vulnerable JS Library	Medium	2
Application Error Disclosure	Low	2
Dangerous JS Functions	Low	4
Permissions Policy Header Not Set	Low	6
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low	11
Timestamp Disclosure - Unix	Low	8
X-Content-Type-Options Header Missing	Low	9
Information Disclosure - Suspicious Comments	Informational	18
Modern Web Application	Informational	1
Non-Storable Content	Informational	2
Storable but Non-Cacheable Content	Informational	9
User Agent Fuzzer	Informational	24

CAPÍTULO II: PROPUESTA

2.1. Fundamentos teóricos aplicados

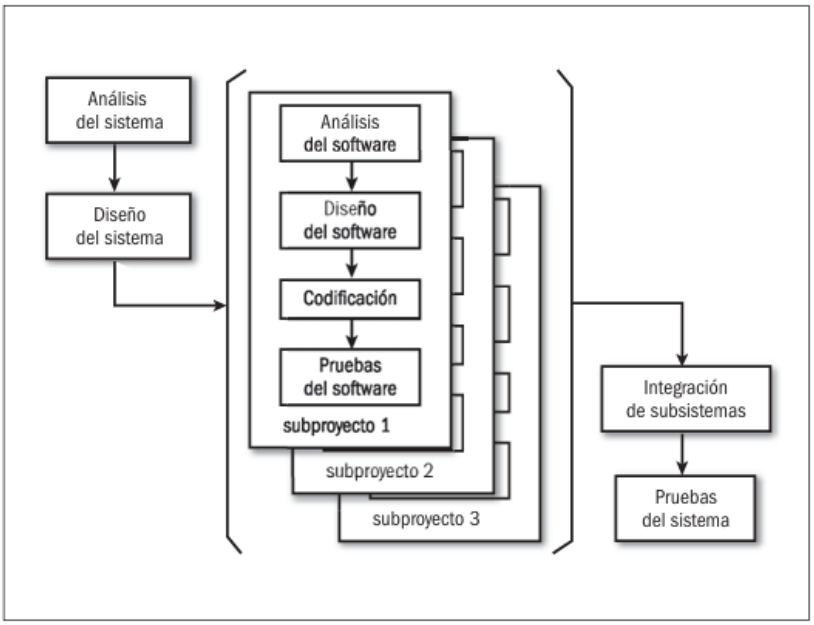
Arquitectura Microservicios

Las arquitecturas basadas en Microservicios están emergiendo actualmente como opciones apropiadas para aplicaciones distribuidas de misión crítica. En una arquitectura basada en microservicios, la aplicación se construye basada en una colección de servicios que deben ser desarrollados, probados, versionados y desplegados en producción, de forma independiente. En paralelo, las empresas actualmente están descubriendo como con contenedores Docker pueden reducir costes, resolver problemáticas de despliegues y en definitiva mejorar DevOps y operaciones de despliegues a producción. Docker se está convirtiendo en un estándar de facto. (De la Torre, 2019).

En la figura 6 se muestra el ciclo de vida de software utilizada en arquitectura de microservicios.

Figura 6.

Ciclo de Vida de un Software Microservicios.



Nota. Ciclo de vida del desarrollo de software (SDLC) (Richard Murch,2019)

Integración continua (CI) y el despliegue continuo (CD)

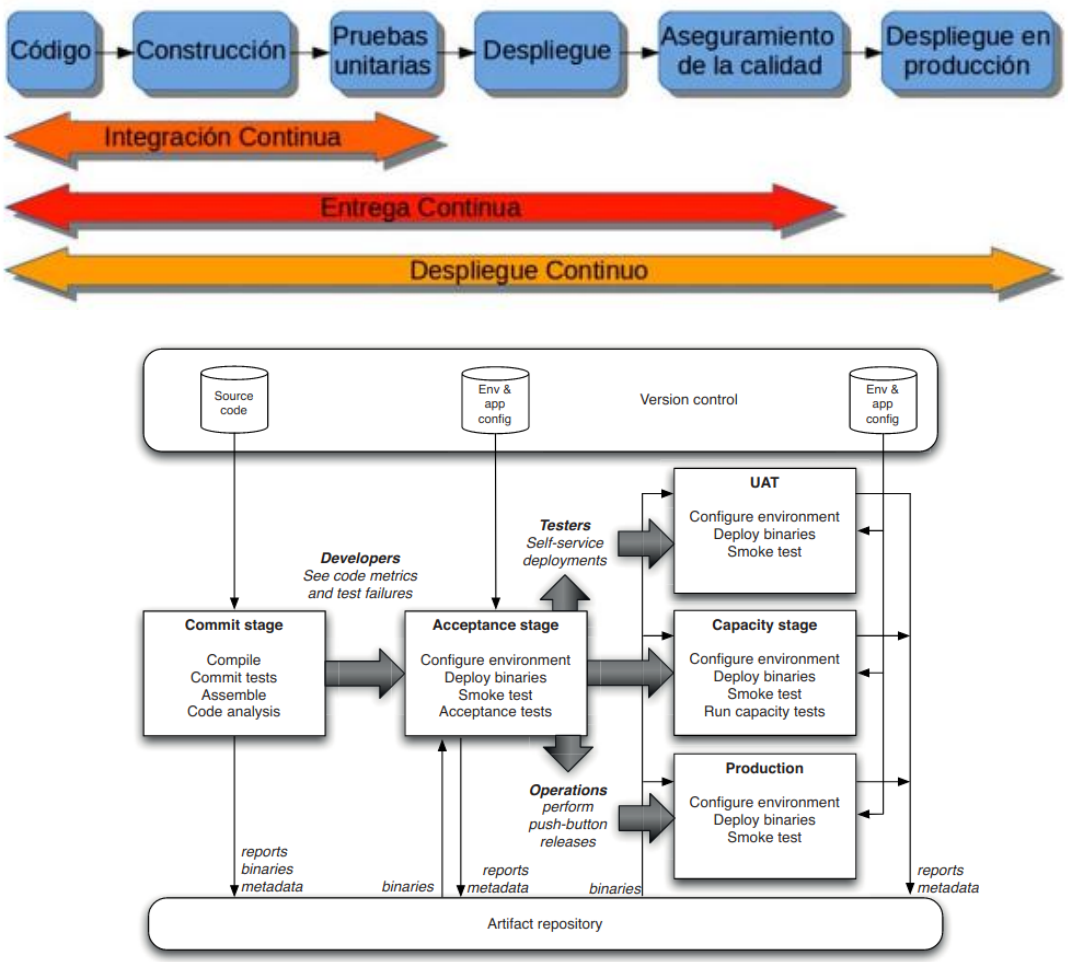
La integración y despliegue continuo son prácticas que han permitido que el ciclo de desarrollo de software se comporte de una manera más ágil, esto gracias a que constantemente

brindan la posibilidad de automatizar procesos con el fin de mejorar tiempos y calidad. Este tipo de prácticas requieren de conocimiento y en la mayoría de los casos algún proveedor que cuente con herramientas y recursos con los cuales se puedan realizar dichas automatizaciones, es por esta razón que las empresas colombianas cuentan con opciones para todo tipo de objetivos y presupuestos económicos (Cruz y Franco, 2022).

La figura 7 detalla el modelo CI/CD utilizado como marco de referencia en DevOps y DevSecOps.

Figura 7.

Esquema Configuración Aplicación.



Nota. Esquema configuración Aplicación creación de una nueva instancia de un pipeline: Continuous Delivery (Jeff Humble, 2011)

DevOps

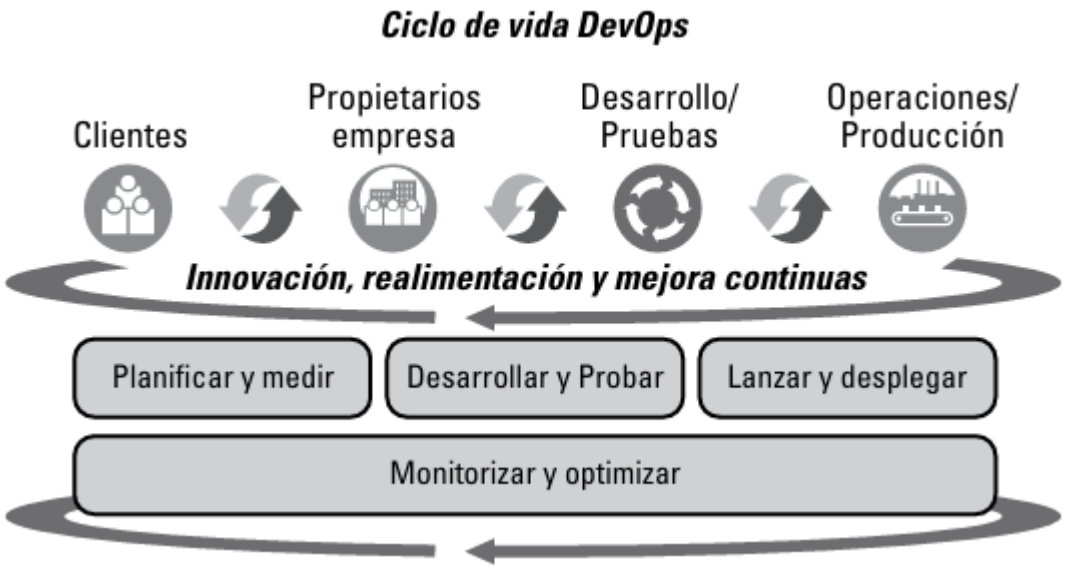
DevOps nace precisamente de la fusión de los equipos de desarrollo y de operaciones con metodologías LEAN y Agile. Los equipos DevOps están formados por miembros multidisciplinares (arquitectos, ingenieros de sistemas, desarrolladores, testeadores,

responsables de producto -producto owners-, entre otros) que son capaces de actuar de forma aislada sin casi ninguna dependencia de otros equipos; disponen del conocimiento y las herramientas necesarias para gestionar el ciclo de vida completo de una aplicación: pueden diseñar una nueva característica, programarla, probarla, generar los entornos necesarios, y desplegar la versión definitiva en producción, mientras evalúan continuamente cómo mejorar y ser más eficientes (Prieto y Cerezo, 2023).

La figura 8 se detalla el ciclo de vida en DevOps.

Figura 8.

Ciclo de vida DevOps



Nota. Arquitectura de Referencia de DevOps. (Sharma, 2014)

La seguridad en el ciclo de vida del software

Actualmente, la seguridad es un requerimiento no funcional que puede ser de nido como la calidad y capacidad del software al momento de enfrentarse a la explotación de las vulnerabilidades causadas por las aplicaciones. Por eso, se han diseñado una serie de metodologías incluidas en el proceso de desarrollo de software seguro, en pro de eliminar o mitigar dichas vulnerabilidades e integrar la seguridad como un componente básico en la arquitectura de cualquier producto de software (Kang y Park, 2018). Existen varias metodologías que establecen una serie de pasos o etapas denominadas ciclo de vida de desarrollo de software seguro (S-SDLC, por sus siglas en ingles), que promueven un software más seguro y capaz de resistir a ataques.

DevSecOps

Si bien los principios de DevSecOps son ampliamente reconocidos, aún existe una importante brecha de investigación en cuanto a la aplicación práctica de las técnicas y principios de DevSecOps por parte de las empresas para mejorar la resiliencia de sus sistemas de software. En particular, se requieren más estudios empíricos que examinen las dificultades, los enfoques y los procedimientos óptimos relacionados con la incorporación de la seguridad en el ciclo de vida de DevOps (Mullangi, 2017). Además, la mayor parte de la literatura publicada actualmente se centra en los aspectos teóricos de DevSecOps, con escasos casos prácticos y consejos útiles para las empresas que desean implementar la técnica.

La combinación de Desarrollo (Dev), Seguridad (Sec) y Operaciones (Ops) se conoce como DevSecOps y supone un cambio de paradigma en la gestión del desarrollo y la seguridad del software por parte de las empresas. DevSecOps es fundamentalmente un movimiento filosófico y cultural que busca mejorar la seguridad y la resiliencia de los sistemas de software mediante la integración fluida de procedimientos de seguridad en el proceso DevOps (Rodríguez et al., 2018).

DevSecOps Maturity Model

Un modelo de madurez de DevSecOps sirve como una guía valiosa para las organizaciones que se esfuerzan por fortalecer sus prácticas de seguridad dentro del desarrollo de software. Al comprender los conceptos del modelo de madurez de DevSecOps, como los contenidos en DSOMM de OWASP, las organizaciones no solo pueden mejorar la seguridad, sino también obtener numerosos beneficios potenciales, desde la reducción de costos hasta la mejora de la experiencia del cliente. DSOMM garantiza que la seguridad se convierta en una parte inherente del proceso de desarrollo al enfatizar la integración de medidas de seguridad en todo el SDLC, similar a las funciones que cumplen las herramientas DevSecOps y las metodologías de pruebas de seguridad. Dentro de una estrategia DevOps, DSOMM fomenta la prueba meticulosa de componentes, como bibliotecas de aplicaciones y bibliotecas del sistema operativo dentro de imágenes de Docker, para identificar y abordar vulnerabilidades conocidas. (Linskens, 2024)

Docker en la seguridad y Automatización del Ciclo de Vida del Software

Los contenedores informáticos están cambiando la forma en que ejecutamos los programas. Aunque facilitan las cosas, necesitamos nuevas maneras de mantenerlos seguros. Las formas antiguas de proteger los sistemas informáticos no funcionan tan bien con los contenedores. Piensa en los contenedores como paquetes sellados que contienen programas. Cada contenedor

tiene sus paredes para mantenerlo separado de otros contenedores. Pero estas paredes necesitan reglas de seguridad especiales para funcionar correctamente.

Los contenedores funcionan de forma más simple al compartir un mismo sistema, lo que los hace más rápidos, pero requiere reglas de seguridad adicionales. Dado que los contenedores comparten más partes del sistema, debemos vigilarlos cuidadosamente y establecer límites claros. Ambas opciones funcionan bien: solo necesitas elegir la que mejor se adapte a tus necesidades.

Automatización de Escaneo de Seguridad

Las protecciones automatizadas integradas en cada fase de desarrollo son la única forma fiable de detectar fallos de seguridad antes de que el software llegue a producción. Tres familias de herramientas son la base de este esfuerzo: SAST (pruebas estáticas de seguridad de aplicaciones), DAST (pruebas dinámicas de seguridad de aplicaciones) y SCA (análisis de composición de software).

SAST escanea el código fuente sin procesar mientras aún se encuentra en la rama del desarrollador, detectando problemas como puntos de inyección o comprobaciones de entrada laxas sin ejecutar el programa. DAST invierte el enfoque, atacando la aplicación en su estado activo para descubrir problemas en tiempo de ejecución: flujos de autenticación débiles, encabezados expuestos u otras fugas de interfaz. SCA completa el trío auditando cada biblioteca y framework de terceros en la compilación, marcando las versiones vinculadas a CVE publicados.

El Proyecto Abierto de Seguridad de Aplicaciones Web destaca el valor de esta combinación, señalando que “las pruebas de seguridad estáticas y dinámicas, combinadas con el análisis de composición de software, ofrecen una cobertura integral para descubrir debilidades en todas las capas del ciclo de desarrollo” (OWASP, 2021, p. 7). Cuando se utilizan juntas, estas técnicas permiten a los equipos de DevSecOps de rápido movimiento reducir los riesgos que conlleva la integración rápida y la entrega ininterrumpida.

2.2. Descripción de la propuesta

La propuesta presenta un marco DevSecOps diseñado para empresas que mantienen sus stacks locales On-Premise. El modelo se desarrolla en cuatro fases vinculadas, cada una de las cuales integra comprobaciones de seguridad que se activan automáticamente a medida que el código avanza por el flujo de entrega. La estructura aborda los puntos críticos comunes en las implementaciones de microservicios: automatización fragmentada, supervisión deficiente y respuesta lenta a vulnerabilidades.

La etapa inicial de diagnóstico y cultrua DevSecOps, describe el estado actual de la situación en cuanto a la seguridad en la organización: cómo los equipos técnicos escriben código, prueban entregan y despliegan software, y cómo consideran el riesgo. Mediante la auditoría de los flujos de trabajo y la cultura, se detectará puntos ciegos, establecer una línea base y unir a los desarrolladores, el personal de operaciones y los responsables de seguridad en torno a un objetivo compartido y automatizado.

La segunda etapa de Diseño y construcción del pipeline seguro. Identifica las utilidades de código abierto que funcionan correctamente en hardware local. Se integra en una nueva pipeline de CI/CD, se define reglas de codificación segura en el (SDLC), planifica la gestión de secretos basada en bóveda y se detalla los pasos para el análisis de código fuente, la verificación de dependencias y la aplicación de controles de calidad desde el primer día.

Una vez finalizado el diseño y contrucción del pipeline seguro, se genera la etapa de implementación. Cada herramienta elegida se instala, se optimiza y se integra en un entorno de pruebas que refleja las cargas de trabajo reales. Aquí se confirma que los análisis SAST, DAST y SCA se inician automáticamente, detectan los problemas con precisión y emiten alertas claras sin intervención humana.

La fase de cierre de Validación, medición y mejora continua, mide lo importante: entradas de registro, informes de análisis, plazos de ejecución de parches y el estado del flujo de trabajo. Las cifras reales muestran dónde la configuración destaca y dónde falla. Los hallazgos se incorporan a los playbooks revisados y a las sesiones de actualización, lo que refuerza un ciclo de mejora continua que mantiene la eficacia del marco mucho después de la finalización del estudio.

a. Estructura general

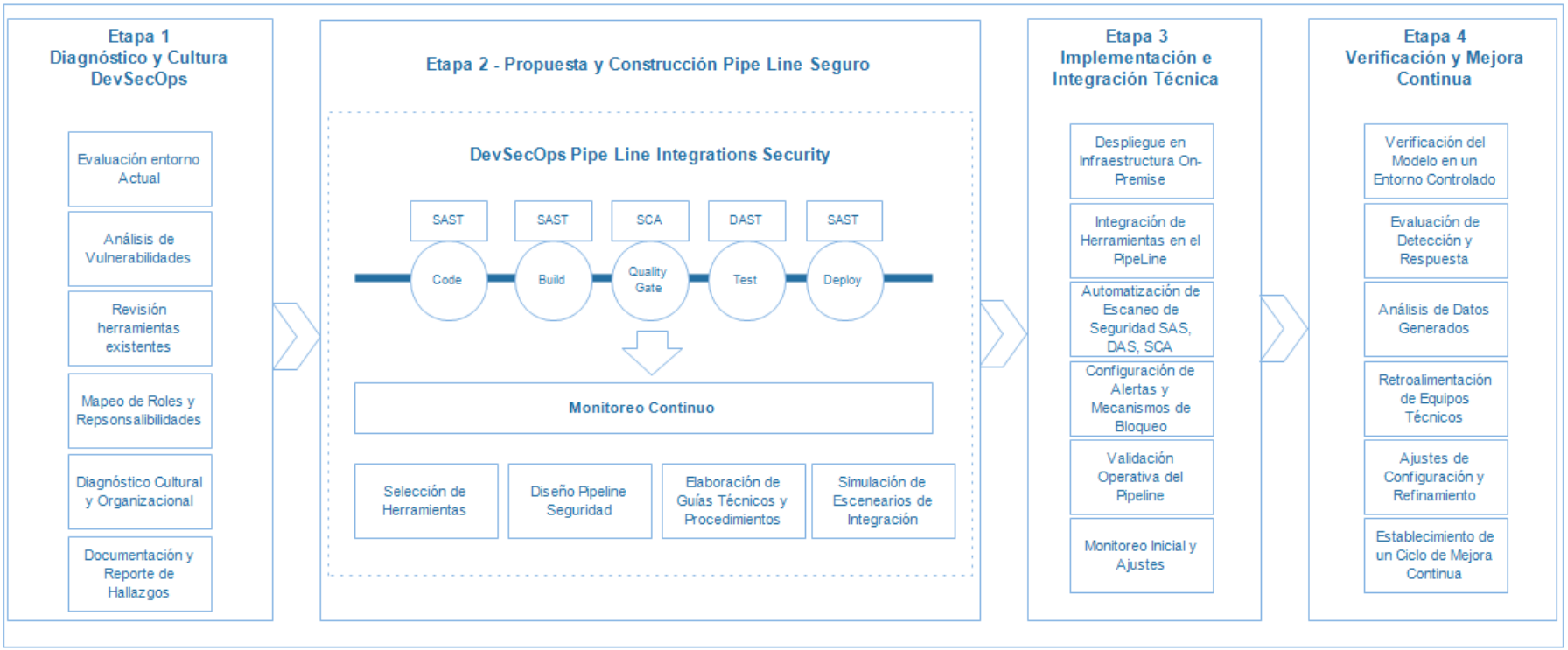
El modelo se presenta como un esquema integral de protección diseñado para infraestructuras on-premise que operan con microservicios. Se divide en cuatro etapas sucesivas, desde un diagnóstico preliminar hasta un ciclo de verificación y ajuste continuo. A lo largo de todo el recorrido se toma como guía DevSecOps y Maturity Model (DSOMM) como referencia que permite medir y aumentar la madurez de las prácticas de seguridad en cada fase, garantizando que las acciones técnicas y organizativas avancen al mismo ritmo que las metas estratégicas de las empresas. Además, la última etapa incorpora el método PDCA (Plan, Do, Check, Act) con el fin de asegurar una retroalimentación constante, mejorar los controles existentes y reaccionar con rapidez ante amenazas emergentes. De este modo, el modelo no solo ofrece una ruta clara para insertar la seguridad en el ciclo de desarrollo, sino que también

aporta un marco metodológico adaptable y escalable que favorece la sostenibilidad, la eficacia y el alineamiento con los principales estándares de buenas prácticas.

La figura 9 muestra el diagrama modelo DevSecOps. Rastrea la seguridad desde la primera confirmación, pasando por la planificación del pipeline y la implementación, hasta una etapa final donde los datos determinan qué funciona y qué necesita ajustes. Cada segmento se integra con el siguiente, dejando claro que las decisiones de diseño iniciales se transmiten a través de la configuración y las pruebas, y que un flujo constante de métricas y retroalimentación debe ser la base del proceso. El diagrama también destaca las transferencias entre equipos, señalando dónde la colaboración es más importante. Junto con las comprobaciones automatizadas de SAST, DAST y SCA integradas en el motor de CI/CD, el modelo consolida una estrategia de seguridad autosostenible que se ajusta a las limitaciones del mundo real y a la tolerancia al riesgo de las organizaciones.

Figura 9.

Organizador gráfico, etapas de la propuesta DevSecOps en arquitectura Microservicios On-Premise. Modelo seguridad.



b. Explicación del aporte

A continuación, se detalla la estructura de la propuesta, que abarca las fases y herramientas que se consideran útiles para resolver la problemática mencionada.

En la tabla 1 se muestra la descripción de la etapa 1 Diagnóstico y Cultura DevSecOps del modelo propuesto DevSecOps.

Tabla 1.

Descripción Etapa 1 Modelo DevSecOps

Proceso	Objetivo	Evidencia Generada
Evaluación del Entorno Actual	Comprender la infraestructura, tecnologías y prácticas actuales.	Informe de entorno actual y puntos débiles.
Análisis de Vulnerabilidades Históricas	Detectar patrones recurrentes de vulnerabilidades y fallos.	Listado de vulnerabilidades históricas clasificadas
Revisión de Herramientas Existentes	Analizar el nivel de automatización y herramientas disponibles.	Matriz de herramientas y nivel de integración
Mapeo de Roles y Responsabilidades	Establecer responsabilidades y distribución de tareas.	Matriz de roles y responsabilidades ajustadas
Diagnóstico Cultural y Organizacional	Evaluar la percepción y disposición organizacional hacia DevSecOps.	Análisis de cultura y barreras organizacionales
Documentación y Reporte de Hallazgos	Documentar hallazgos clave y sentar las bases para las siguientes etapas.	Informe técnico con prioridades identificadas

La tabla 2 muestra el detalla de cada proceso de la etapa 2 Propuesta y Construcción del modelo DevSecOps con pipeline seguro.

Tabla 2.

Descripción Etapa 2 modelo DevSecOps

Proceso	Objetivo	Resultado Esperado
Selección de Herramientas Open Source	Evaluar y seleccionar herramientas libres para CI/CD, análisis de seguridad y gestión de secretos.	Listado de herramientas aprobadas y documentadas con sus especificaciones técnicas.
Definición de Buenas Prácticas	Establecer guías y lineamientos de seguridad para todo el ciclo de vida del software.	Manual de buenas prácticas DevSecOps adaptado al entorno de la organización.
Diseño del Pipeline CI/CD	Definir la arquitectura técnica y los flujos de trabajo del pipeline.	Diagrama y documentación técnica del pipeline CI/CD con integraciones de seguridad.

En la tabla 3 se muestra la descripción de la etapa 3 Integración e implementación Técnica de la propuesta DevSecOps

Tabla 3.

Descripción Etapa 3 modelo DevSecOps

Proceso	Objetivo	Resultado Esperado
Despliegue en Infraestructura On-Premise	Implementar el modelo DevSecOps en el entorno físico de la organización	Entorno funcional con el modelo implementado
Integración de Herramientas en el Pipeline	Conectar las herramientas de seguridad y CI/CD para flujo automatizado	Pipeline funcional con herramientas integradas
Automatización de Escaneo de Seguridad SAST, DAST, SCA	Configurar y ejecutar análisis automáticos de seguridad en el pipeline	Reportes automáticos de vulnerabilidades
Configuración de Alertas y Mecanismos de Bloqueo	Establecer alertas y bloqueos automáticos ante vulnerabilidades críticas	Alertas y bloqueos configurados
Validación Operativa del Pipeline	Verificar la funcionalidad y seguridad del pipeline en producción controlada	Pipeline validado y documentado
Monitoreo Inicial y Ajustes	Observar el funcionamiento y realizar ajustes correctivos iniciales	Pipeline optimizado tras ajustes iniciales

En la tabla 4 se muestra la descripción de la etapa 4 Verificación y Mejora Continua del modelo propuesto DevSecOps.

Tabla 4.

Descripción Etapa 3 modelo DevSecOps

Proceso	Objetivo	Resultado Esperado
Verificación del Modelo en un Entorno Controlado	Evaluar la funcionalidad del modelo en un ambiente aislado	Informe de verificación del modelo
Evaluación de Detección y Respuesta	Medir tiempos de detección y efectividad de la respuesta ante incidentes	Reporte de evaluación de incidentes
Análisis de Datos Generados	Interpretar la información de escaneos, alertas y métricas de seguridad	Informe analítico con métricas clave
Retroalimentación de Equipos Técnicos	Recoger sugerencias y experiencias de los equipos involucrados	Lista de mejoras y observaciones

Ajustes de Configuración y Refinamiento	Optimizar la configuración y procedimientos según hallazgos	Modelo ajustado y optimizado
Establecimiento de un Ciclo de Mejora Continua	Definir procedimientos de revisión y actualización constante	Plan de mejora continua implementado

c. Estrategias y/o técnicas

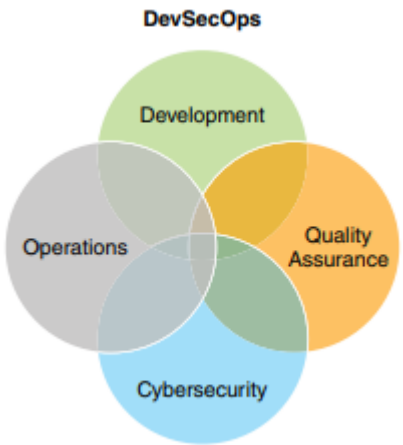
DevSecOps ofrece la capacidad de entregar productos y servicios más seguros al mercado de forma rápida. Durante décadas, los ingenieros de tecnología han buscado equilibrar la velocidad de entrega con la seguridad y el rendimiento. DevSecOps altera fundamentalmente esta ecuación, permitiendo a las empresas entregar con rapidez sin comprometer la seguridad, la privacidad o el rendimiento del sistema. (MACK, 2024).

En el modelo propuesto, se utiliza como estrategia el marco DevSecOps, se aplica desde la fase de planificación, se exige la definición de políticas de seguridad como código y su versionado que alberga la aplicación; durante la construcción, los pipelines ejecutan SAST, SCA y análisis de secretos en cada confirmación; en la etapa de prueba, las imágenes de contenedor pasan por escaneos Trivy y pruebas DAST orquestadas por Jenkins; y, en la fase final, se habilita monitoreo con Prometheus y alertas de cumplimiento sobre métricas de riesgo. Cada paso registra evidencias, de modo que la organización conozca su nivel de madurez y traces objetivos de mejora. Así, el marco DevSecOps no solo inserta controles técnicos, sino que establece un circuito continuo de medición, aprendizaje y ajuste.

En la figura 10 se muestra un esquema general del marco DevSecOps.

Figura 10.

DevSecOps intersección entre operaciones, desarrollo, aseguramiento de la calidad y ciberseguridad



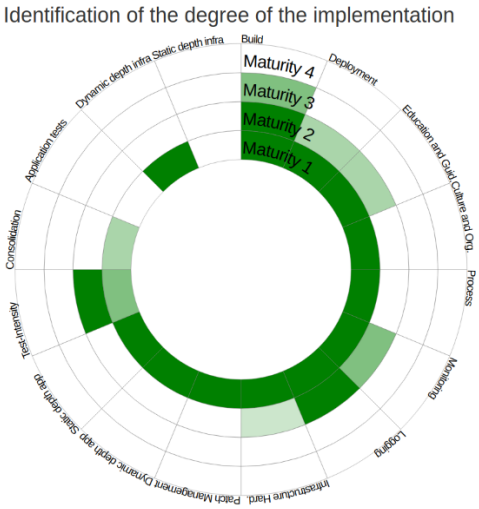
El marco DSOMM consta de cuatro niveles de madurez de DevSecOps. Cada nivel representa una etapa distinta en la evolución de la integración de la seguridad, desde la concientización básica y las prácticas ad hoc hasta los procesos de seguridad avanzados, totalmente integrados y automatizados. Estos niveles proporcionan una hoja de ruta para que las organizaciones mejoren sistemáticamente su estrategia de seguridad dentro del marco DevOps (Katz, 2024).

El marco DSOMM se referencia en el modelo propuesto en todas las fases, funciona como base para valorar la madurez de la seguridad incorporada. Durante el diagnóstico inicial, facilita la lectura del panorama actual de controles y prácticas; en el diseño del modelo, respalda la fijación de metas y lineamientos de mejora; a lo largo de la implantación y la integración técnica, guía la adopción gradual de herramientas y procesos coherentes con las mejores referencias del sector; y, en la fase de verificación y perfeccionamiento continuo, ofrece métricas para cuantificar avances, descubrir vacíos y definir ajustes correctivos. Con esta dinámica, se garantiza que el nivel de protección crezca de forma sostenida y en sincronía con el ciclo de vida del software que se ejecuta sobre infraestructura local.

En la figura 11 se presenta la implementación del marco de referencia DSOMM.

Figura 11.

DSOMM Nivel de Implementación.



Nota. Modelo de madurez DSOMM, (OWASP, 2024)

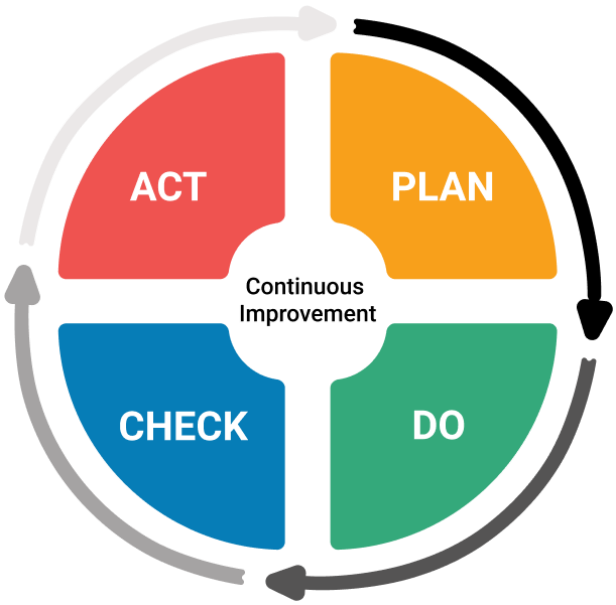
PDCA o Ciclo de Deming es una metodología de gestión que tiene como objetivo la mejora constante de los procesos. Este ciclo consta de cuatro pasos: planificar (plan), hacer (do), verificar (check) y actuar (act). Para adaptarse a los cambios del mercado, aumentar la eficacia, acelerar la productividad y satisfacer las necesidades de los clientes es necesario tener un método (Sydle, 2023).

PDCA se utiliza como técnica en la última fase del modelo, verificación y mejora continua, que impulsa la evolución permanente de la seguridad. También se toma como referencia como se planifican acciones de optimización basadas en las métricas reunidas; se ejecutan ajustes en procesos, herramientas y configuraciones; se verifica su eficacia mediante auditorías, revisión de registros y pruebas de vulnerabilidad; y, finalmente, se actúa corrigiendo desviaciones y trasladando las lecciones aprendidas a la documentación y a la cultura corporativa. Así, el PDCA sostiene un flujo constante de retroalimentación y convierte la seguridad en una práctica dinámica, sostenible y ajustable a las particularidades de los entornos on-premise.

La figura 12 muestra el ciclo de PDCA.

Figura 12.

Ciclo PDCA.



Nota. Ciclo planificar, hacer, revisar y actuar (Naydenov, 2018)

2.3. Validación de la propuesta

La validación del modelo propuesto se realizó por tres especialistas expertos en seguridad informática, seleccionados por su trayectoria académica y experiencia práctica con seguridad local en arquitecturas microservicios y procesos de integración y despliegue continuo. Con experiencia en informática, operaciones de centros de datos y diseño de pipelines seguros, analizaron el plan en profundidad. La tabla 5 muestra información sobre la titulación académica, cargo y años de experiencia de los especialistas que realizan la validación de la propuesta.

Tabla 5.

Descripción perfiles especialistas.

Nombres y Apellidos	Años de Experiencia	Titulación Académica	Cargo
Jonathan Marcelo Díaz Almachi	8 años	Magister en Seguridad Informática.	Docente Instituto Universitario Cordillera, Consultor externo, Ciberseguridad - Servife
Lino Jesús Cajas Pacheco	7 años	Magister en Seguridad Informática	Docente Instituto Universitario Cordillera, Consultor externo
Edwin Alexander Alobuela Muñoz	7 años	Ingeniero en sistemas informáticos.	Analista de TI - Aliservis SA - Domino's Ecuador, Consultor Externo

El resultado de la revisión, abarcó los puntos donde la seguridad se integra en cada paso de CI/CD y la relevancia de las tareas asignadas a cada fase de implementación. Las observaciones se registraron en una plantilla estructurada y se transformaron en ajustes específicos que reforzaron tanto la practicidad como el rigor técnico, manteniendo el modelo alineado con los estándares vigentes. Los especialistas también revisaros la fase de prueba (registros de compilación, informes de escáner y métricas de tiempo) para verificar las mejoras declaradas e identificar los puntos donde una protección u optimización adicional aportaría más valor.

2.4. Matriz de articulación de la propuesta

La tabla 6 muestra la matriz articulación de la propuesta del modelo DevSecOps

Tabla 6.
Matriz de articulación

EJES O PARTES PRINCIPALES	SUSTENTO TEÓRICO	SUSTENTO METODOLÓGICO	ESTRATEGIAS / TÉCNICAS	DESCRIPCIÓN DE RESULTADOS	INSTRUMENTOS APLICADOS
Etapa 1 Diagnóstico y Cultura DevSecOps	DevSecOps, la evolución natural de DevOps, un enfoque que combina desarrollo operaciones y seguridad. (Gonzáles, 2024)	Metodología bibliográfica.	Fuente bibliográficas, Marco de Referencia DevSecOps	Permite Identificar el estado actual de seguridad, inventario de herramientas, mapeo de roles y cultura organizacional frente a DevSecOps.	Lista de verificación de vulnerabilidades, observación
Etapa 2 Propuesta y construcción del pipeline seguro	Fundamento en principios de CI/CD con seguridad embebida, siguiendo guías de OWASP SAMM y DevSecOps Maturity Model.	Investigación Exploratoria y Descriptiva	Fuente bibliográficas, Marco de Referencia DevSecOps Maturity Model y OWASP	Permite Desarrollar la propuesta de pipeline para la integración segura adaptado a microservicios On-Premise y documentación técnica de procesos.	Análisis de contenido, scripts de integración, diagramas de pipeline

Etapa 3 Integración e implementación Técnica	Metodologías ágiles para la implementación de infraestructura segura y automatización de controles de seguridad.	Investigación Descriptiva, Observación Directa	Fuente bibliográficas, Marco de Referencia DevSecOps Maturity Model y OWASP	Permite la validación del Pipeline en funcionamiento, herramientas integradas, alertas y bloqueos activos, monitoreo inicial de seguridad.	Análisis de contenido, herramientas de escaneo automatizado, sistema de alertas, panel de monitoreo
Etapa 4 Verificación y Mejora Continua	Basada en enfoques de mejora continua como PDCA y métricas de seguridad	Observación directa	Fuente bibliográficas, Marco de Referencia DevSecOps Maturity Model y OWASP	Permite validar la efectividad del modelo, mejoras aplicadas y ciclo de mejora continua establecido.	Análisis de contenido, Reportes de validación, métricas de desempeño, actas de retroalimentación, plan de mejoras.

CONCLUSIONES

La investigación confirmó que las vulnerabilidades predominantes en arquitecturas de microservicios alojadas on-premise se originan, principalmente, en la gestión inadecuada de secretos, fallos de autenticación y autorización, configuraciones inseguras de contenedores y la falta de controles automatizados en los pipelines de integración y despliegue continuos. Estos hallazgos subrayan la necesidad de un programa de seguridad holístico que articule salvaguardas técnicas con políticas organizativas.

La propuesta establece un conjunto de lineamientos y buenas prácticas que facilita la adopción de DevSecOps en instalaciones locales: integración temprana de escaneos SAST, DAST y SCA; orquestación CI/CD con gestión centralizada de secretos; monitoreo continuo; y un marco de roles, capacitación y métricas que consolida una cultura de seguridad compartida.

Se diseñó un modelo de seguridad cimentado en DevSecOps que encadena controles automáticos en cada etapa del pipeline CI/CD: escaneos SAST, DAST y SCA, gestión de secretos y monitoreo continuo. La solución detecta y mitiga vulnerabilidades de forma temprana sin frenar la entrega de microservicios on-premise.

El modelo se validó por especialistas dentro de un laboratorio on-premise que replicó cargas y fallos reales; los evaluadores corroboraron la eficacia de los controles, validaron la fiabilidad de las alertas y, tras sugerir ajustes menores, confirmaron la viabilidad operativa de la propuesta.

RECOMENDACIONES

Se recomienda que todas las organizaciones que adopten el modelo consoliden la corresponsabilidad entre desarrollo, operaciones y seguridad. Para ello conviene formalizar políticas internas precisas, impartir programas de formación recurrentes y desplegar iniciativas de sensibilización que aseguren la presencia de controles de seguridad en cada fase del ciclo de vida del software.

Se recomienda incorporar marcos de referencia consolidados como OWASP, NIST o la norma ISO/IEC 27001 como complemento al modelo. Estos estándares ofrecen directrices probadas que refuerzan la gestión segura de vulnerabilidades, configuraciones e infraestructura en entornos locales.

Conviene incorporar de forma continua herramientas de SAST, DAST, SCA, gestión de secretos y monitoreo en las canalizaciones CI/CD, de modo que los escaneos se ejecuten en cada fase de integración y despliegue. Asimismo, resulta prudente revisar periódicamente el rendimiento y la cobertura de estas utilidades para mantener la vigencia y exactitud de sus hallazgos.

Se recomienda que se realice pilotos del modelo mediante entornos de prueba controlados y, solo tras su validación, lo extienda a producción. Esta adopción escalonada permite afinar la configuración, mitigar riesgos y asegurar una transición sólida hacia un pipeline seguro.

BIBLIOGRAFÍA

Benedictis, A. (2022). DevSecOps: Integrating Security into DevOps Lifecycle. O'Reilly Media.

Dias, N., & Siriwardena, P. (2020). Microservices Security in Action: Design secure microservice architecture. Manning Publications.

Fernández-Alemán, J. L., Carrillo de Gea, J. M., & Toval, A. (2020). La importancia de integrar la seguridad en el ciclo de vida del desarrollo de software. Revista Española de Informática y Seguridad, 17(1), 5–12.

International Organization for Standardization & International Electrotechnical Commission. (2018). ISO/IEC 27001:2018 — Information technology — Security techniques — Information security management systems — Requirements. ISO. https://www.iso.org/standard/54534.html

Kim, G., Humble, J., Debois, P., & Willis, J. (2016). The DevOps handbook: How to create world-class agility, reliability, & security in technology organizations. IT Revolution Press.

Klijnstra, S. (2024). DevSecOps practices in CI/CD pipelines (Master's thesis). Universiteit Leiden, Países Bajos.

Mack, S. D. (2023). The DevSecOps Playbook: Deliver Continuous Security at Speed. John Wiley & Sons.

Mendoza Obregón, J. J., & Leyva Guadalupe, M. A. (2024). Propuesta de un marco integral de DevSecOps para mitigar la fuga de información en el ciclo de vida del desarrollo de software en una empresa de seguros. Universidad Peruana de Ciencias Aplicadas, Perú. Repositorio UPC-Institucional.

Moyón Constante, F., Soares, R., Pinto-Albuquerque, M., Méndez, D., & Beckers, K. (2021). Integration of Security Standards in DevOps Pipelines: An Industry Case Study. arXiv preprint. https://arxiv.org/abs/2105.13024

National Institute of Standards and Technology. (2020). NIST Special Publication 800-53 Revision 5: Security and Privacy Controls for Information Systems and Organizations. U.S. Department of Commerce. https://doi.org/10.6028/NIST.SP.800-53r5

OWASP Foundation. (2021). OWASP Top 10 – 2021: The Ten Most Critical Web Application Security Risks. https://owasp.org/Top10/

Rajapakse, R. N., Zahedi, M., Babar, M. A., & Shen, H. (2021). Challenges and solutions when adopting DevSecOps: A systematic review. arXiv preprint. https://arxiv.org/abs/2103.08266

Shirm, G., Humble, J., Forsgren, N., & Willis, J. (2018). Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations. IT Revolution Press.

ANEXOS

ANEXO 1

VALIDACIÓN ESPECIALISTAS

Universidad Israel
ESPOG | Escuela de Posgrados

UNIVERSIDAD TECNOLÓGICA ISRAEL

ESCUELA DE POSGRADOS "ESPOG"

MAESTRÍA EN SEGURIDAD INFORMÁTICA

INSTRUMENTO PARA VALIDACIÓN DE LA PROPUESTA

Estimado colega:

Se solicita su valiosa cooperación para evaluar la calidad del siguiente contenido digital "Propuesta de un modelo de seguridad DevSecOps para detección y mitigación automatizada de vulnerabilidades en arquitecturas de microservicios On-Premise". Sus criterios son de suma importancia para la realización de este trabajo, por lo que se le pide que brinde su cooperación contestando las preguntas que se realizan a continuación.

Datos informativos

Validado por: Jonathan Marcelo Díaz Almachi
Título obtenido: Magister en Seguridad Informática
C.I.: 1722467923
E-mail: jonathan.diaz@cordillera.edu.ec
Institución de Trabajo: Instituto Superior Tecnológico Cordillera, Servife
Cargo: Docente, Consultor externo Ciberseguridad
Años de experiencia en el área: 9 años

Instructivo:

- Responda cada criterio con la máxima sinceridad del caso.
- Revisar, observar y analizar la propuesta de la plataforma virtual, blog o sitio web.
- Coloque una X en cada indicador, tomando en cuenta que Muy adecuado equivale a 5, Bastante Adecuado equivale a 4, Adecuado equivale a 3, Poco Adecuado equivale a 2 e Inadecuado equivale a 1.

Página **1** de **2**

Universidad Israel

ESPOG | Escuela de Posgrados

Tema: "Propuesta de un modelo de seguridad DevSecOps para detección y mitigación automatizada de vulnerabilidades en arquitecturas de microservicios On-Premise"

Indicadores	Muy adecuado	Bastante Adecuado	Adecuado	Poco adecuado	Inadecuado
Pertinencia	X				
Aplicabilidad	X				
Factibilidad	X				
Novedad		X			
Fundamentación pedagógica		X			
Fundamentación tecnológica	X				
Indicaciones para su uso	X				
TOTAL	**25**	**8**			

Observaciones: El modelo demuestra un alto nivel de integración entre herramientas de análisis, control de versiones, monitoreo y gestión de vulnerabilidades, lo cual garantiza un pipeline robusto y seguro. Esta integración refuerza la trazabilidad en cada fase del ciclo de vida del software, permitiendo detectar y mitigar riesgos de manera proactiva.

Recomendaciones: Se recomienda a las organizaciones que adopten el modelo propuesto invertir en un plan de sensibilización y capacitación continua dirigido a los equipos de desarrollo, operaciones y seguridad. La cultura DevSecOps exige colaboración permanente y responsabilidad compartida, por lo que alinear a los equipos desde el inicio resulta fundamental para garantizar la efectividad de la propuesta en la práctica.

Lugar, fecha de validación: Quito D.M., agosto 2025

Firma del especialista
Jonathan Marcelo Díaz Almachi

UNIVERSIDAD TECNOLÓGICA ISRAEL

ESCUELA DE POSGRADOS "ESPOG"

MAESTRÍA EN SEGURIDAD INFORMÁTICA

INSTRUMENTO PARA VALIDACIÓN DE LA PROPUESTA

Estimado colega:

Se solicita su valiosa cooperación para evaluar la calidad del siguiente contenido digital "Propuesta de un modelo de seguridad DevSecOps para detección y mitigación automatizada de vulnerabilidades en arquitecturas de microservicios On-Premise". Sus criterios son de suma importancia para la realización de este trabajo, por lo que se le pide que brinde su cooperación contestando las preguntas que se realizan a continuación.

Datos informativos

Validado por: Lino Jesús Cajas Pacheco
Título obtenido: Magister en Seguridad Informática
C.I.: 1721061214
E-mail: lino.cajas@cordillera.edu.ec
Institución de Trabajo: Instituto Superior Tecnológico Cordillera
Cargo: Docente, Consultor externo Ciberseguridad
Años de experiencia en el área: 10 años

Instructivo:

- Responda cada criterio con la máxima sinceridad del caso.
- Revisar, observar y analizar la propuesta de la plataforma virtual, blog o sitio web.
- Coloque una X en cada indicador, tomando en cuenta que Muy adecuado equivale a 5, Bastante Adecuado equivale a 4, Adecuado equivale a 3, Poco Adecuado equivale a 2 e Inadecuado equivale a 1.

Tema: "Propuesta de un modelo de seguridad DevSecOps para detección y mitigación automatizada de vulnerabilidades en arquitecturas de microservicios On-Premise"

Indicadores	Muy adecuado	Bastante Adecuado	Adecuado	Poco adecuado	Inadecuado
Pertinencia	X				
Aplicabilidad	X				
Factibilidad	X				
Novedad	X				
Fundamentación pedagógica		X			
Fundamentación tecnológica	X				
Indicaciones para su uso	X				
TOTAL	**30**	**4**			

Observaciones: El modelo propuesto no solo mejora aspectos técnicos, sino que también fomenta la colaboración efectiva entre los equipos de desarrollo, operaciones y seguridad. Al fortalecer la cultura organizacional hacia un enfoque DevSecOps, se potencia la capacidad de respuesta frente a incidentes y se impulsa una visión compartida de responsabilidad en la seguridad del software.

Recomendaciones: Se recomienda seleccionar herramientas alineadas al ecosistema de la organización, priorizando aquellas que sean compatibles con entornos On-Premise y que cuenten con comunidades activas. Jenkins, SonarQube, OWASP ZAP y Trivy constituyen ejemplos sólidos, pero es clave validar periódicamente su efectividad y realizar actualizaciones para mantenerse al día frente a nuevas amenazas.

Lugar, fecha de validación: Quito D.M., agosto 2025

Firma del especialista
Lino Jesús Cajas Pacheco

UNIVERSIDAD TECNOLÓGICA ISRAEL

ESCUELA DE POSGRADOS "ESPOG"

MAESTRÍA EN SEGURIDAD INFORMÁTICA

INSTRUMENTO PARA VALIDACIÓN DE LA PROPUESTA

Estimado colega:

Se solicita su valiosa cooperación para evaluar la calidad del siguiente contenido digital "Propuesta de un modelo de seguridad DevSecOps para detección y mitigación automatizada de vulnerabilidades en arquitecturas de microservicios On-Premise". Sus criterios son de suma importancia para la realización de este trabajo, por lo que se le pide que brinde su cooperación contestando las preguntas que se realizan a continuación.

Datos informativos

Validado por: Edwin Alexander Alobuela Muñoz
Título obtenido: Ingeniero en sistemas informáticos
C.I.: 1721353678
E-mail: ealobuela@dominos.com.ec
Institución de Trabajo: Aliservis S.A. Domino's Ecuador
Cargo: Analista de TI.
Años de experiencia en el área: 9 años

Instructivo:

- Responda cada criterio con la máxima sinceridad del caso.
- Revisar, observar y analizar la propuesta de la plataforma virtual, blog o sitio web.
- Coloque una X en cada indicador, tomando en cuenta que Muy adecuado equivale a 5, Bastante Adecuado equivale a 4, Adecuado equivale a 3, Poco Adecuado equivale a 2 e Inadecuado equivale a 1.

Universidad Israel

ESPOG | Escuela de Posgrados

Tema: "Propuesta de un modelo de seguridad DevSecOps para detección y mitigación automatizada de vulnerabilidades en arquitecturas de microservicios On-Premise"

Indicadores	Muy adecuado	Bastante Adecuado	Adecuado	Poco adecuado	Inadecuado
Pertinencia	X				
Aplicabilidad	X				
Factibilidad	X				
Novedad		X			
Fundamentación pedagógica		X			
Fundamentación tecnológica	X				
Indicaciones para su uso	X				
TOTAL	**25**	**8**			

Observaciones: El modelo propuesto integra la seguridad en las fases del ciclo de vida del software de manera transversal, lo cual representa un cambio cultural y técnico importante respecto a los enfoques tradicionales donde la seguridad se aplicaba al final del proceso. Esta visión anticipada permite detectar y mitigar riesgos en etapas tempranas, reduciendo costos y aumentando la resiliencia del sistema frente a vulnerabilidades.

Recomendaciones: Se recomienda establecer un proceso continuo de actualización y revisión de las herramientas utilizadas en el pipeline, dado que las soluciones de seguridad evolucionan de forma constante. Esto asegurará que el modelo no pierda vigencia con el tiempo y que siempre cuente con versiones mejoradas para detección y mitigación de riesgos.

Lugar, fecha de validación: Quito D.M., agosto 2025

Firma del especialista
Edwin Alexander Alobuela Muñoz

ANEXO 2

ESTRUCTURA DEL MODELO

Universidad Israel

MODELO DE SEGURIDAD DEVSECOPS PARA DETECCIÓN Y MITIGACIÓN AUTOMATIZADA DE VULNERABILIDADES EN ARQUITECTURAS DE MICROSERVICIOS ON-PREMISE

Autor: Stalin Mauricio Mejía M.

2025

Contenido

Índice de Tablas

Índice de Figuras

Introducción

El modelo DevSecOps aquí descrito sirve como hoja de ruta para integrar la seguridad de forma sistemática en entornos on-premise basados en microservicios. Su propósito consiste en ofrecer a las organizaciones un esquema práctico y ordenado que refuerce la protección de las aplicaciones desde las fases iniciales del ciclo de vida, en consonancia con el enfoque shift-left y con marcos de madurez como DevSecOps Maturity Model (DSOMM).

La propuesta parte de la premisa de que los métodos clásicos de desarrollo y operaciones requieren una evolución que incluya controles de seguridad continuos y automáticos dentro de los pipelines de integración y entrega continua. Con este fin, el modelo recomienda el empleo de herramientas de código abierto garantizando viabilidad técnica y costes asumibles para infraestructuras locales.

Además de los aspectos tecnológicos, el planteamiento incorpora la dimensión cultural y organizativa, al fomentar la cooperación entre los equipos de desarrollo, operaciones y seguridad. De este modo, cada grupo asume responsabilidades claras sobre el software y la infraestructura subyacente, promoviendo una cultura de protección compartida.

El documento adjunto expone el modelo en cuatro fases: diagnóstico y cultura DevSecOps, diseño y construcción, despliegue e integración técnica y verificación y mejora continua. Cada fase describe procesos, metas, herramientas y entregables definidos, configurando un plan de trabajo claro, medible y adaptable a diversos contextos empresariales.

Descripción del Modelo de Seguridad Informática

El modelo se presenta como un esquema integral de protección diseñado para infraestructuras on-premise que operan con microservicios. Se divide en cuatro etapas sucesivas, desde un diagnóstico preliminar hasta un ciclo de verificación y ajuste continuo. A lo largo de todo el recorrido se toma como guía DevSecOps y Maturity Model (DSOMM) como referencia que permite medir y aumentar la madurez de las prácticas de seguridad en cada fase, garantizando que las acciones técnicas y organizativas avancen al mismo ritmo que las metas estratégicas de las empresas. Además, la última etapa incorpora el método PDCA (Plan, Do, Check, Act) con el fin de asegurar una retroalimentación constante, mejorar los controles existentes y reaccionar con rapidez ante amenazas emergentes. De este modo, el modelo no solo ofrece una ruta clara para insertar la seguridad en el ciclo de desarrollo, sino que también aporta un marco metodológico adaptable y escalable que favorece la sostenibilidad, la eficacia y el alineamiento con los principales estándares de buenas prácticas.

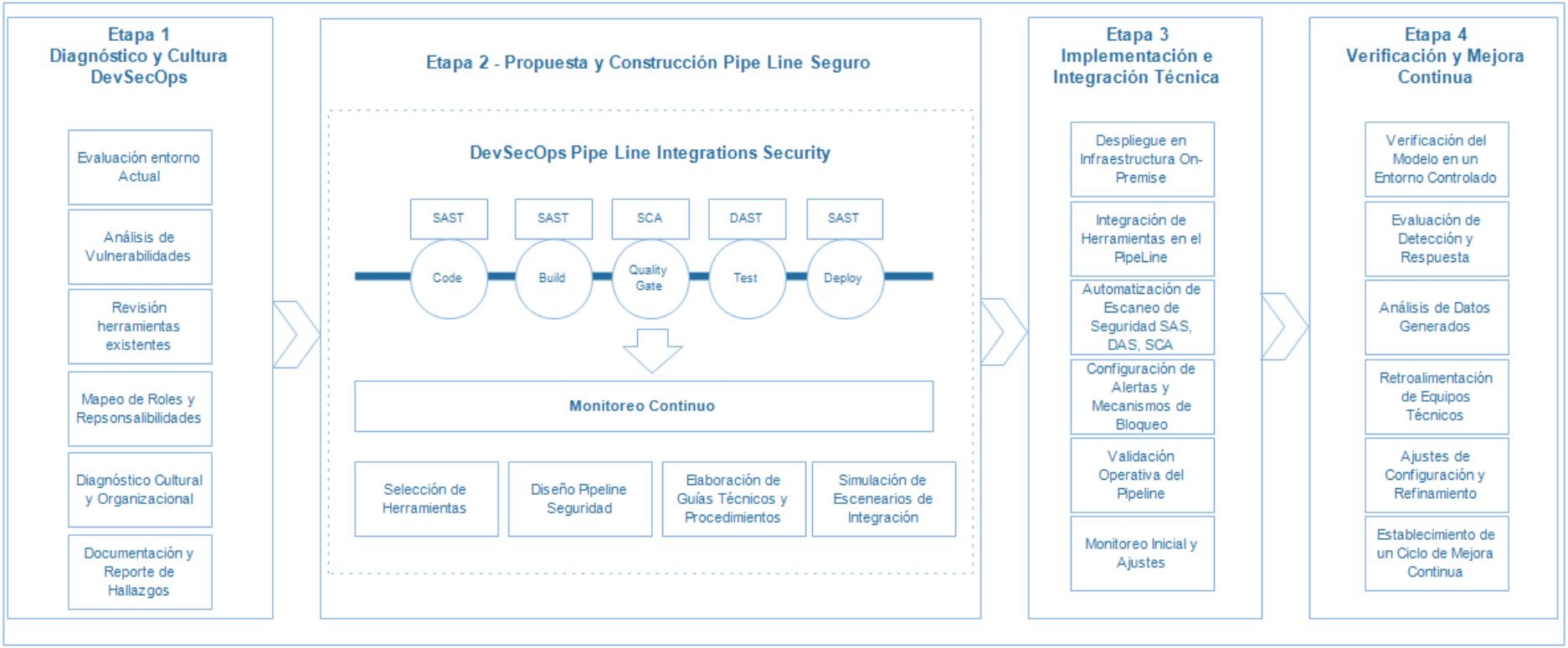
1. Elementos del Modelo

1.1 Diagrama del modelo

La figura 1 muestra modelo DevSecOps propuesto para el análisis y mitigación de vulnerabilidades en entornos On-Premise con arquitectura microservicios.

Figura 1.

Etapas del modelo DevSecOps



1.2 Principales componentes de seguridad.

El modelo combina recursos técnicos y organizativos que, de forma complementaria, refuerzan la seguridad a lo largo del ciclo de vida del software dentro de un marco DevSecOps. En la parte técnica incorpora prácticas de análisis estático, dinámico y de composición (SAST, DAST y SCA), administración centralizada de secretos, supervisión continua y escaneos automáticos integrados en la canalización CI/CD; todo ello facilita la detección temprana de fallos y su corrección ágil. De manera paralela, la dimensión organizativa contempla la asignación clara de funciones, el diagnóstico cultural, la formación del personal, con el fin de fomentar una cultura colaborativa y consciente en materia de seguridad. Así, los mecanismos técnicos proporcionan la protección operativa y tecnológica requerida, mientras que los elementos organizacionales establecen la estructura cultural y de gobernanza imprescindible para que la incorporación de la seguridad resulte sostenible y eficaz durante cada fase del desarrollo.

En la tabla 1 se muestra los componentes principales de seguridad del modelo DevSecOps propuesto.

Tabla 1.

Componentes Principales modelo DevSecOps

Etapa	Componentes Técnicos	Componentes Organizacionales
Etapa 1: Diagnóstico y Cultura DevSecOps	Revisión de infraestructura y procesos actuales, Análisis de vulnerabilidades históricas, Inventario de herramientas existentes, Documentación técnica inicial	Mapa de roles y responsabilidades, Diagnóstico cultural y organizacional, Sensibilización de equipos, Reporte de hallazgos
Etapa 2: Propuesta y Construcción	Pipelines CI/CD con seguridad integrada, Herramientas SAST, DAST y SCA, Gestión de secretos y credenciales, Simulación de escenarios de integración	Selección de herramientas, Diseño de guías y procedimientos técnicos, Elaboración de lineamientos metodológicos
Etapa 3: Implementación e Integración Técnica	Infraestructura On-Premise, Integración de herramientas en el pipeline, Automatización de escaneos, Alertas y mecanismos de bloqueo, Validación del pipeline, Monitoreo inicial	Gestión de equipos DevSecOps, Coordinación entre desarrollo, operaciones y seguridad, Retroalimentación en la validación
Etapa 4: Verificación y Mejora Continua	Verificación en entorno controlado, Evaluación de detección y respuesta, Análisis de datos generados, Ajustes de configuración y refinamiento, Establecimiento de ciclo de mejora continua	Retroalimentación de equipos técnicos, Cultura de mejora continua, Documentación de lecciones aprendidas

2. Implementación del Modelo DevSecOps

La puesta en marcha del modelo debe realizarse de forma gradual y ordenada, iniciando con un diagnóstico preliminar del contexto tecnológico y organizacional para reconocer fortalezas y debilidades.

2.1 Pasos aplicación Etapa 1 - Diagnóstico y Cultura DevSecOps

En la tabla 2 se muestra los pasos de implementación en la fase 1 del modelo propuesto.

Tabla 2.

Pasos Etapa 1 - Diagnóstico y Cultura DevSecOps.

No.	Proceso	Pasos de Implementación	Instrumentos	Resultado Esperado
1	Evaluación del Entorno Actual	Levantar información de arquitecturas y flujos actuales. Revisar prácticas de seguridad vigentes. Identificar puntos críticos sin controles.	Diagramas de arquitectura, checklist de seguridad, entrevistas técnicas.	Informe de evaluación del entorno y mapeo de riesgos iniciales.
2	Análisis de Vulnerabilidades Históricas	Recopilar registros de incidentes previos. Clasificar vulnerabilidades más frecuentes. Identificar brechas recurrentes.	Logs históricos, reportes de incidentes, bases de datos CVE.	Reporte de vulnerabilidades históricas categorizadas por tipo y severidad.
3	Revisión de Herramientas Existentes	Inventariar herramientas CI/CD, testing y despliegue. Evaluar nivel de automatización. Verificar existencia de controles de seguridad integrados.	Inventario de software, auditoría técnica.	Lista de herramientas actuales con su nivel de madurez y brechas de seguridad.
4	Mapeo de Roles y Responsabilidades	Identificar actores clave (Dev, Ops, QA, Sec). Definir roles preliminares para DevSecOps.	Entrevistas técnicas, organigrama empresarial.	Mapa de roles y responsabilidades inicial para adopción de DevSecOps.

5	Diagnóstico Cultural y Organizacional	Realizar focus groups o talleres. Identificar resistencias al cambio. Evaluar nivel de colaboración entre áreas.	Focus groups, cuestionarios internos, observación participativa.	Reporte de cultura organizacional y nivel de madurez cultural hacia DevSecOps.

En la fase inicial de despliegue del modelo DevSecOps resulta imprescindible combinar entrevistas, listas de verificación, registros e inventarios con el apoyo de marcos y normas internacionales como ISO, OWASP, que unifican los criterios de evaluación. De este modo, los entregables generados: mapa de riesgos, informe de vulnerabilidades, asignación de roles y diagnóstico cultural, mantienen fiabilidad, comparabilidad y coherencia con las buenas prácticas, y proporcionan un cimiento sólido para las etapas siguientes del modelo.

En la tabla 3 se lista marcos y normas de referencia para cada proceso de la etapa 1 como una recomendación para un resultado estandarizado.

Tabla 3.

Marcos y normas de referencia recomendados para los procesos de la etapa 1.

Proceso	Instrumentos Aplicados	Recomendación para Uso Estandarizado	Resultado Fortalecido
1. Evaluación del Entorno Actual	Diagramas de arquitectura, checklist de seguridad, entrevistas técnicas	Utilizar marcos como ISO/IEC 27001 e ISO/IEC 27002 para evaluar controles de seguridad y guías de arquitectura de referencia.	Informe más preciso del entorno y mapa inicial de riesgos.
2. Análisis de Vulnerabilidades Históricas	Logs históricos, reportes de incidentes, bases CVE	Basarse en estándares como NIST CVSS para la clasificación de vulnerabilidades y guías de OWASP.	Reporte de vulnerabilidades categorizado y priorizado por impacto y severidad.
3. Revisión de Herramientas Existentes	Inventario de software, auditoría técnica	Referenciar buenas prácticas de CIS Controls y comparar con estándares de madurez como OWASP SAMM o DSOMM.	Lista de herramientas con su nivel de madurez y brechas de seguridad identificadas.

4. Mapeo de Roles y Responsabilidades	Entrevistas técnicas, organigrama empresarial	Seguir lineamientos de ISO/IEC 38500 (gobernanza de TI) y Matriz Raci para definir funciones claras en DevSecOps.	Roles y responsabilidades definidos para iniciar la adopción.
5. Diagnóstico Cultural y Organizacional	Focus groups, cuestionarios internos, observación participativa	Apoyarse en metodologías de gestión del cambio como ADKAR y prácticas de ISO 27005 para medir madurez organizacional en seguridad.	Reporte de cultura y nivel de madurez hacia DevSecOps.

2.2 Pasos aplicación Etapa 2 - Propuesta y construcción

2.2.1 Selección de Herramientas

La elección de herramientas durante la fase de Diseño y Construcción del modelo representa un paso decisivo; con ella se asegura que los procesos de integración y de seguridad dentro de la cadena DevSecOps respondan a las exigencias del entorno corporativo. En esta etapa se comparan alternativas técnicas según criterios tales como compatibilidad con microservicios, facilidad de acoplamiento a pipelines CI/CD, respaldo comunitario o comercial, costos y grado de automatización alcanzable.

Planificación.

En esta etapa se debe elegir una metodología para el desarrollo de microservicios en una arquitectura distribuida que sea compatible con el modelo DevSecOps.

La tabla 4 describe una comparación de las metodologías más utilizadas en entornos On-Premise para desarrollo de microservicios en modelos DevSecOps.

Tabla 4.

Metodologías para planificación de desarrollo de software en arquitecturas microservicios.

Metodología	Descripción	Ventajas en microservicios	Limitaciones
Ágil (Agile)	Conjunto de principios que promueven entregas iterativas, colaboración cercana con el cliente y adaptación constante al cambio.	Flexibilidad ante cambios de requisitos, entregas incrementales y mejora continua; base cultural de metodologías como Scrum y Kanban.	Puede carecer de estructura clara si no se adapta correctamente; riesgo de pérdida de documentación formal.
Scrum	Marco ágil basado en ciclos cortos de trabajo (sprints), con roles definidos (Product Owner, Scrum Master, Equipo de Desarrollo).	Permite priorizar entregables frecuentes, facilita la adaptación al cambio y mejora la colaboración entre equipos.	Puede generar sobrecarga de reuniones si no se gestiona bien; menos flexible para proyectos con requisitos poco definidos.
Kanban	Sistema visual de gestión de tareas mediante tableros con columnas (pendiente, en proceso, terminado).	Excelente para gestionar flujos continuos, alta visibilidad del trabajo y fácil adaptación de prioridades.	No define roles ni plazos estrictos, lo que puede complicar proyectos complejos si no hay disciplina.

La metodología más utilizada para el desarrollo de microservicios en entorno On-Premise en Scrum, utiliza un marco basado en ciclos y ayuda a la realizar entregas en tiempos cortos, la vinculación con los modelos DevOps y DevSecOps permite realizar entregas incrementales.

Desarrollo

En la fase de desarrollo se tiene que aplicar el uso de herramientas para tener un control de versiones, para rastrear y gestionar los cambios realizados en un proyecto de software orientado hacia microservicios.

La tabla 5 realiza una comparación de las herramientas utilizadas para control de versiones en proyectos de software.

Tabla 5.

Herramientas para control de versiones en el desarrollo de software.

Sistema	Características	Licenciamiento	Aplicabilidad en microservicios	Ventajas destacadas
Git (plataformas como GitHub, GitLab, Bitbucket)	Distribuido, cada usuario tiene copia completa del repositorio. Amplio soporte de ramas y fusión.	(open source). Servicios como GitHub/GitLab ofrecen planes de pago con funciones adicionales (CI/CD, seguridad, etc.).	Muy adecuado: facilita desarrollo descentralizado de múltiples microservicios en repos separados o monorepos.	Popularidad, integración con CI/CD, soporte masivo en la industria.
Subversion (SVN)	Centralizado. Todo se gestiona desde un servidor central.	Open source gratuito, algunos proveedores ofrecen planes pagos de hosting.	Aplicable, pero menos flexible para microservicios distribuidos.	Estructura clara, buena gestión de permisos en repositorios centralizados.
Mercurial (Hg)	Distribuido, similar a Git, pero con sintaxis más sencilla.	Gratuito (open source).	Compatible, pero menos adoptado en la industria, lo que limita su ecosistema.	Buena gestión de ramas, rendimiento en repos grandes.
Perforce (Helix Core)	Centralizado/distribuido híbrido. Optimizado para grandes equipos y repositorios enormes (ej. videojuegos, empresas grandes).	Licencias comerciales, con versión gratuita para equipos pequeños (hasta 5 usuarios).	Muy útil en proyectos grandes de microservicios con repositorios pesados y alta concurrencia.	Escalabilidad, seguridad avanzada, buen soporte empresarial.

La herramienta más utilizada para el control de versiones en arquitectura microservicios en entorno On-Premise es Git, por el desarrollo descentralizado ayudando a los desarrolladores a colaborar, deshacer cambios y mantener un historial de todo el proyecto de software.

Análisis de Secretos

El escaneo de secretos constituye una práctica de seguridad orientada a localizar, administrar y resguardar credenciales sensibles; contraseñas, tokens, claves de API o certificados que con frecuencia aparecen expuestas en el código fuente, archivos de configuración o repositorios. Su objetivo es reducir el riesgo de fugas capaces de comprometer la infraestructura, las aplicaciones y los datos corporativos.

La tabla 6 muestra nombre y descripción de las herramientas utilizadas para resguardar credenciales sensibles compatibles con desarrollo en aplicaciones con microservicios para su ejecución en el modelo DevSecOps, con el fin de ayudar a seleccionar la más adecuada.

Tabla 6.

Herramientas para análisis de secretos.

Nombre de la herramienta	Descripción	Web de referencia
HashiCorp Vault	Plataforma open source para almacenar y gestionar secretos con control de acceso y rotación automática.	https://www.vaultproject.io
CyberArk Conjur	Herramienta para gestionar secretos en pipelines CI/CD y contenedores, compatible con Kubernetes.	https://www.conjur.org
Doppler	Plataforma SaaS que simplifica la gestión de variables de entorno y sincronización de secretos.	https://www.doppler.com
Infisical	Herramienta open source y SaaS para centralizar secretos en proyectos y pipelines CI/CD.	https://infisical.com
AWS Secrets Manager	Servicio de Amazon para guardar y rotar credenciales de manera automática en entornos AWS.	https://aws.amazon.com/secrets-manager
Azure Key Vault	Solución de Microsoft para proteger claves, secretos y certificados en aplicaciones y servicios Azure.	https://azure.microsoft.com/services/key-vault

Google Secret Manager	Servicio de Google Cloud para almacenar y administrar secretos con control de acceso basado en IAM.	https://cloud.google.com/secret-manager

Infisical al ser una herramienta opensource, es una de las mejores alternativas, sin embargo, la determinación final se basa en las necesidades del proyecto y preferencias del grupo.

Servidor de Integración continua y entrega continua (CI/CD), Automatización.

Los servidores de CI/CD y las utilidades de automatización ocupan un lugar decisivo en la eficiencia, la seguridad y la repetibilidad de los procesos de despliegue. Gracias a estas plataformas se encadenan tareas críticas de forma automática: compilación de código, ejecución de pruebas, escaneos de seguridad, despliegues controlados y verificación de calidad, lo que reduce la intervención humana y, por ende, el margen de error y exposición. Por este motivo, la elección acertada de dichas herramientas resulta determinante dentro del planteamiento y construcción del modelo DevSecOps, pues fija el grado de madurez tecnológica, la facilidad de integración y la capacidad de asegurar entregas consistentes y protegidas en infraestructuras locales.

En el cuadro comparativo de la tabla 7 se mencionan varias herramientas de servidores CI/CD compatibles con arquitecturas locales. Jenkins es una de las mejores opciones debido a que es open source, maneja pipelines declarativos, mantiene un gran ecosistema de plugins para herramientas de terceros como Docker para el modelo propuesto.

Tabla 7.

Herramientas servidore CI/CD para automatización de procesos de integración y entrega continua.

Herramienta	Características	Web de referencia
Jenkins	Servidor CI/CD open source, muy extendido; pipelines declarativos; gran ecosistema de plugins; integración con Docker y Kubernetes.	https://www.jenkins.io
GitLab CI/CD (Self-Managed)	Integrado con repositorio y control de versiones; runners on-prem; registro de contenedores; gestión completa en una sola plataforma.	https://about.gitlab.com

Argo CD	GitOps para Kubernetes; sincronización declarativa; rollbacks automáticos; muy usado en despliegues on-prem de microservicios.	https://argo-cd.readthedocs.io
Ansible	Automatización de configuración y despliegues; playbooks YAML; no requiere agentes; flexible para infraestructura y apps.	https://www.ansible.com
Rundeck	Orquestación de tareas operativas y runbooks; permite delegar operaciones; control de acceso y auditoría integrados.	https://www.rundeck.com

Pruebas estáticas de seguridad de aplicaciones (SAST)

El análisis SAST constituye un recurso eficaz para localizar vulnerabilidades habituales, entre ellas fallos de codificación, inyecciones SQL o Cross-Site Scripting (XSS). De ese modo permite al equipo de desarrollo corregir los riesgos de seguridad antes de publicar el producto. Para efectuarlo se recurre a utilidades especializadas que recorren el código fuente, línea a línea, y buscan patrones asociados a debilidades conocidas. Dichas utilidades pueden acoplarse al flujo DevSecOps, de forma que el examen se ejecute de manera continua mientras el software evoluciona. No obstante, es recomendable complementar todo el proceso con revisiones manuales periódicas que descarten posibles falsos positivos y destapen eventuales incidencias que la herramienta automática por sí sola no consiga identificar con precisión.

La tabla 8 presenta las herramientas utilizadas para el escaneo de vulnerabilidades mediante pruebas estáticas SAST, una de las alternativas más recomendada por la comunidad es SonarQube, en sus versiones opensource al poseer un catálogo amplio de lenguajes de programación con Javascript, C#, Java, Python entre otros.

Tabla 8.

Herramientas SAST para escaneo de vulnerabilidades

Nombre de la herramienta	Descripción	Web de referencia
SonarQube	Plataforma de análisis de código que permite identificar vulnerabilidades, malas prácticas y errores de calidad en múltiples lenguajes de programación. Se integra fácilmente en pipelines CI/CD.	https://www.sonarsource.com/products/sonarqube/
Checkmarx SAST	Solución comercial que ofrece análisis profundo de código fuente para detectar vulnerabilidades de seguridad en fases tempranas del ciclo de desarrollo.	https://checkmarx.com/
Fortify Static Code Analyzer (SCA)	Herramienta de Micro Focus que analiza el código fuente y proporciona reportes detallados sobre vulnerabilidades y cumplimiento normativo.	https://www.microfocus.com/en-us/cyberres/application-security/static-code-analyzer
Veracode SAST	Plataforma en la nube que realiza análisis estático de aplicaciones, brindando reportes detallados y priorización de riesgos de seguridad.	https://www.veracode.com/
Codacy	Herramienta SaaS de revisión de código que soporta múltiples lenguajes, detecta vulnerabilidades, problemas de estilo y métricas de calidad.	https://www.codacy.com/
Bandit (Python)	Herramienta open source especializada en el análisis de seguridad de proyectos en Python, enfocada en detectar vulnerabilidades comunes en librerías y dependencias.	https://bandit.readthedocs.io/
Semgrep	Analizador estático de código ligero y altamente configurable que permite crear reglas personalizadas para detectar vulnerabilidades en diversos lenguajes.	https://semgrep.dev/

PMD	Herramienta open source que revisa código en Java y otros lenguajes, detectando errores de programación, vulnerabilidades y problemas de estilo.	https://pmd.github.io/

Análisis de Componentes de software (SCA)

El análisis de componentes de software consiste en escanear las bibliotecas y dependencias de terceros incluidas en un proyecto para reconocer vulnerabilidades registradas o debilidades presentes en la versión empleada. Los resultados se cotejan con catálogos públicos de fallos conocidos a fin de establecer el nivel de riesgo. Cuando se confirma una deficiencia, se procede a mitigarla, ya sea actualizando el componente a una edición corregida o sustituyéndolo por otra opción más segura.

Esta práctica resulta crítica dentro del modelo DevSecOps, porque los atacantes suelen preferir explotar vulnerabilidades en dependencias externas antes que abordar directamente el código propio de la aplicación. Al ejecutar de manera sistemática un análisis de composición efectivo, el equipo detecta y corrige con rapidez las vulnerabilidades presentes en componentes aportados por terceros, lo que refuerza de forma notable la postura global de seguridad de la solución. Además, el proceso genera informes trazables que facilitan auditorías y brindan visibilidad a las partes interesadas.

La tabla 9 muestra herramientas SCA para escaneo de dependencias, se evalúan 8 herramientas, con la finalidad de especificar la opción más adecuada. La herramienta Owasp al utilizar una base de datos NVD puede considerarse factible para entornos on-premise y compatible con entornos microservicios, por otra parte, la herramienta Trivy además permite el escaneo de contenedores a nivel de sistemas operativo y muestra informes de seguridad con códigos CVE, serían una gran alternativa ya que los microservicios en su mayoría llegan a convertirse en instancias de contenedores.

Tabla 9.

Herramientas SCA para escaneo de dependencias de terceros.

Nombre de la herramienta	Descripción	Web de referencia
OWASP Dependency-Check	Herramienta open source que analiza dependencias en proyectos de software y detecta vulnerabilidades conocidas mediante bases como NVD (National Vulnerability Database).	https://owasp.org/www-project-dependency-check/
Snyk	Plataforma comercial y open source que identifica vulnerabilidades en dependencias, contenedores y configuraciones, con integración en CI/CD.	https://snyk.io/
WhiteSource (ahora Mend)	Solución de SCA que gestiona riesgos de licencias y vulnerabilidades de código abierto, con reportes en tiempo real.	https://www.mend.io/
Sonatype Nexus Lifecycle	Ofrece un análisis profundo de componentes open source, detectando vulnerabilidades y riesgos de cumplimiento en todo el ciclo de vida del software.	https://www.sonatype.com/products/nexus-lifecycle
Black Duck (Synopsys)	Una de las herramientas líderes en SCA, utilizada para gestionar vulnerabilidades, licencias y riesgos de seguridad en componentes de terceros.	https://www.synopsys.com/software-integrity/software-composition-analysis.html
FOSSA	Plataforma de SCA que automatiza la detección de vulnerabilidades y riesgos de licencias en código abierto.	https://fossa.com/

Anchore	Especializada en análisis de imágenes de contenedores, detecta vulnerabilidades en las dependencias incluidas en los paquetes y capas del contenedor.	https://anchore.com/
Trivy (Aqua Security)	Escáner de seguridad open source. Analiza imágenes Docker, dependencias, archivos IaC (Infrastructure as Code) y repositorios, detectando vulnerabilidades, configuraciones débiles y secretos expuestos. Ideal para integrarlo en pipelines DevSecOps.	https://aquasecurity.github.io/trivy/

Pruebas dinámicas de seguridad de aplicaciones (DAST)

Para ejecutar pruebas DAST se recurre a utilidades capaces de emular ataques contra la aplicación; dichas utilidades lanzan peticiones dirigidas al servicio y luego inspeccionan las respuestas devueltas. Con ellas se aplican múltiples estrategias para destapar fallos, entre las que se encuentran la inyección de datos, la fuerza bruta o la suplantación de identidad. Cuando el motor detecta una debilidad, genera un informe con la ubicación exacta, una explicación y la recomendación pertinente para subsanarla.

Estas evaluaciones se catalogan como pruebas de “caja negra”, dado que se parte del supuesto de que el evaluador no conoce ni puede manipular el funcionamiento interno de la aplicación; en otras palabras, no tiene acceso al código en ejecución ni a los procesos de depuración. Por lo general, las pruebas se ejecutan de manera automática mediante plataformas especializadas, aunque también se complementan con verificaciones manuales efectuadas por pentesters o auditores de seguridad.

En la tabla 10 se describe una comparativa de las herramientas que se utilizan para el escaneo de pruebas dinámicas en seguridad de aplicaciones. Owasp Zap es una alternativa compatible con entornos On-Premise y procesos CI CD, debido a que esta herramienta cuenta con integración y capacida de análisis y la comunidad de usuarios que la soporta.

Tabla 10.

Herramientas DAST para pruebas dinámicas de seguridad en aplicaciones.

Nombre de la herramienta	Descripción	Web de referencia
OWASP ZAP (Zed Attack Proxy)	Herramienta open source desarrollada por OWASP para realizar pruebas dinámicas de seguridad en aplicaciones web. Permite descubrir vulnerabilidades como inyección SQL, XSS y fallos de configuración.	https://www.zaproxy.org/
Burp Suite	Plataforma comercial muy utilizada para pruebas de seguridad en aplicaciones web. Su versión profesional permite automatizar escaneos dinámicos de seguridad y realizar pruebas manuales avanzadas.	https://portswigger.net/burp
Acunetix	Escáner de seguridad web automatizado que detecta vulnerabilidades en aplicaciones web, incluyendo OWASP Top 10, y proporciona reportes priorizados.	https://www.acunetix.com/
Netsparker (Invicti)	Herramienta de análisis dinámico que identifica vulnerabilidades en aplicaciones web y APIs,	https://www.invicti.com/
AppScan (HCL Security)	Solución comercial que ejecuta pruebas dinámicas y ofrece integración con CI/CD para asegurar aplicaciones durante el ciclo de desarrollo.	https://www.hcltechsw.com/appscan
W3af (Web Application Attack and Audit Framework)	Framework open source enfocado en la identificación de vulnerabilidades web como XSS, inyección de SQL, CSRF, y más.	http://w3af.org/
Nikto	Escáner open source que revisa servidores web en busca de configuraciones inseguras, software obsoleto o vulnerabilidades comunes.	https://cirt.net/Nikto2

Monitoreo

El monitoreo, considerado al elegir las herramientas durante la definición y el ensamblaje del modelo DevSecOps, constituye un factor decisivo, pues permite vigilar en tiempo real el rendimiento, la solidez y la seguridad del pipeline como de los microservicios desplegados. Su valor reside en la visibilidad que aporta sobre indicadores clave, consumo de recursos, salud de contenedores, flujo de red, detección de anomalías y avisos de seguridad, lo cual favorece la detección temprana de incidentes y vulnerabilidades.

La tabla 11 realiza una comparación con 8 herramientas que existen para el monitoreo de varios factores como rendimiento, seguridad, indicadores clave con la finalidad de elegir una que sea compatible con el modelo propuesto. Prometheus y Grafana se presentan con las mejores opciones al tener integración en procesos CI CD con arquitectura en micro servicios, soportan varios lenguajes y tipos de comunicación para el monitoreo en tiempo real de los servicios.

Tabla 11.

Herramientas para monitoreo en tiempo real compatibles con DevSecOps.

Herramienta de Monitoreo	Tipo	Lenguaje Soportado	Precio Mensual
Prometheus	Sistema de Monitoreo de métricas	Go	Gratis
Zabbix	Sistema de Monitoreo integral	Varios	Gratis / De pago
Nagios	Sistema de Monitoreo de infraestructura	Varios	Gratis
Grafana	Análisis y visualización de datos	Varios	Gratis
Datadog	Plataforma de observabilidad	Varios	De pago
ELK Stack	Plataforma de monitoreo y análisis de logs	Varios	Gratis / De pago (Elastic Cloud)
New Relic	Observabilidad y monitoreo de aplicaciones	Varios	De pago
Splunk	Monitoreo, análisis y correlación de datos	Varios	De pago

Gestión de vulnerabilidades

La gestión de vulnerabilidades constituye un eje fundamental al escoger las herramientas que sustentarán la propuesta y el desarrollo de un modelo DevSecOps, pues facilita la detección, clasificación, priorización y corrección de debilidades presentes en sistemas, aplicaciones y componentes dentro de una arquitectura de microservicios. Su meta consiste en disminuir la superficie de ataque y asegurar un tratamiento proactivo de dichas fallas antes de que terceros puedan explotarlas.

Por último, en la tabla 12 se realiza una presentación sobre las herramientas para la detección de vulnerabilidades sustenta a la propuesta del modelo DevSecOps, OWASP ZAP representa una opción válida ya que utiliza un proxy que intercepta y analiza el tráfico permitiendo identificar vulnerabilidades de seguridad utilizando una variedad de técnicas y complementos basado en el marco de referencia OWASP.

Tabla 12.

Herramientas para la gestión de vulnerabilidades

Nombre de la herramienta	Descripción	Web de referencia
Anchore Engine	Analiza vulnerabilidades en imágenes Docker.	anchore.com
Clair	Escáner estático de vulnerabilidades en contenedores.	github.com/quay/clair
GVM (Greenbone Vulnerability Management)	Framework de gestión de vulnerabilidades con escaneo automatizado.	greenbone.net
Lynis	Auditoría de seguridad para sistemas Unix/Linux.	cisofy.com/lynis
OpenVAS	Escáner de vulnerabilidades de código abierto, parte de GVM.	openvas.org
OSSEC	Sistema de detección de intrusos basado en host (HIDS).	ossec.net

OWASP ZAP	Escáner para pruebas de seguridad en aplicaciones web.	owasp.org
Security Onion	Plataforma para monitoreo, detección de intrusos y análisis forense.	securityonionsolutions.com
Snort	Sistema de detección y prevención de intrusos basado en red (NIDS/NIPS).	snort.org
Wazuh	Plataforma de monitoreo de seguridad y cumplimiento normativo.	wazuh.com

Diseño Pipeline Seguridad

Todas estas herramientas se adaptan sin problemas a infraestructuras locales y resultan idóneas para levantar un pipeline seguro en escenarios de microservicios. La tabla 13 muestra el papel que desempeña cada pieza a lo largo del ciclo de desarrollo: GitLab actúa como sistema de control de versiones; Jenkins coordina las tareas de CI/CD; Infisical administra secretos; SonarQube junto con OWASP Dependency-Check revisan código y librerías, mientras OWASP ZAP ejecuta pruebas dinámicas y gestiona hallazgos. Por último, Prometheus y Grafana ofrecen monitoreo continuo y visibilidad operativa.

Tabla 13.

Herramientas para el diseño del pipeline Seguro.

Herramienta	Tipo	Rol en el Pipeline Seguro
Jenkins	Servidor CI/CD	Orquesta la construcción, pruebas y despliegue automatizado de microservicios.
GitLab	Control de Versiones	Gestiona el código fuente, control de cambios y colaboración entre equipos.
Infisical	Análisis de Secretos	Protege credenciales y llaves, asegurando la gestión de secretos en todo el ciclo de vida.

SonarQube	SAST (Static Application Security Testing)	Analiza el código fuente para identificar vulnerabilidades y malas prácticas.
Travy	SCA (Software Composition Analysis)	Evalúa las dependencias externas para detectar vulnerabilidades conocidas.
OWASP ZAP	DAST (Dynamic Application Security Testing) / Gestión de Vulnerabilidades	Realiza pruebas dinámicas simulando ataques y apoya en la identificación de vulnerabilidades.
Prometheus y Grafana	Monitoreo	Detectan anomalías en tiempo real, recopilan métricas y permiten la visualización gráfica.

Construcción del Pipeline para DevSecOps

Para la construcción de un pipeline seguro necesitamos un entorno de simulación o laboratorio, donde se implementa las herramientas seleccionadas para la ejecución del modelo propuesto DevSecOps.

Entorno Simulación

Servidor Docker en Debian

Para la creación de un entorno controlado, la figura 2 detalla la configuración inicial de la instalación de Docker en un servidor Debian 12, se utiliza el método de instalación con los repositorios oficiales.

Figura 2.

Comandos oficiales para la instalación de Docker en una infraestructura local en Linux con Debian.

```
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.ascA
```

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/debian \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

```
#Instalar ultima version de Docker
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

#Verificar Instalacion
sudo docker run hello-world
```

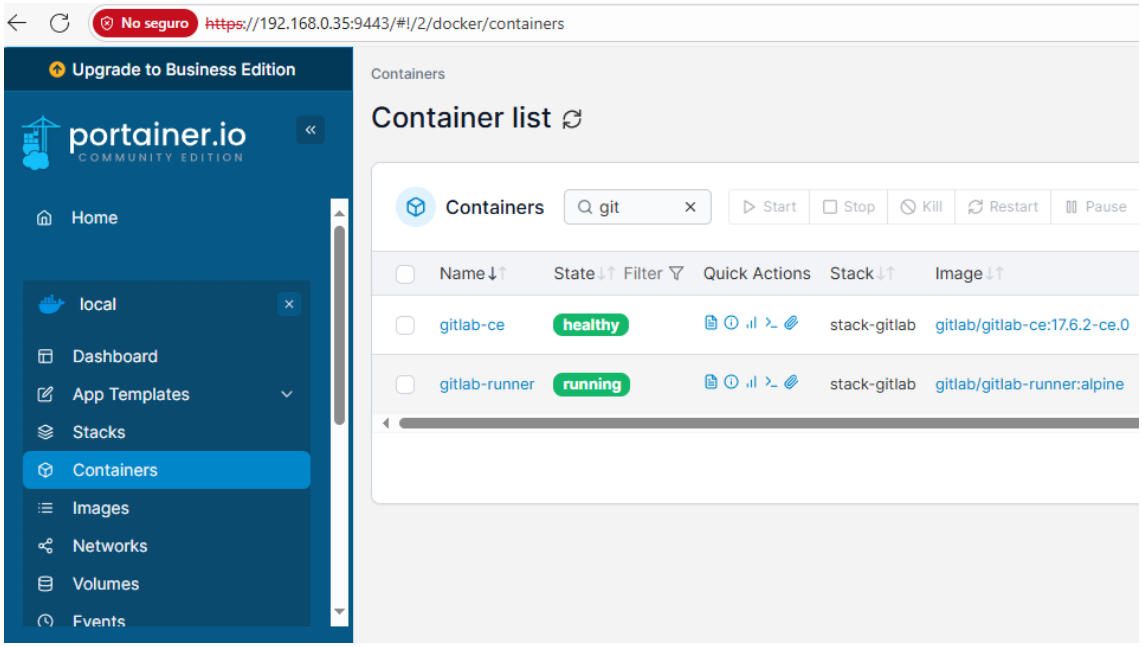
```
it@srvuiodocker02: ~
it@srvuiodocker02:~$ docker --version
Docker version 25.0.4, build 1a576c5
it@srvuiodocker02:~$ docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED         STATUS
                                                                  NAMES
e6f328c9df6b   gitlab/gitlab-runner:alpine    "/usr/bin/dumb-init …"   8 months ago    Up 5 weeks
                                                                  gitlab-runner
e3111bfafa7c   gitlab/gitlab-ce:17.6.2-ce.0   "/assets/wrapper"        8 months ago    Up 5 weeks (healthy)
```

Para gestionar las imágenes de Docker mediante una interfaz gráfica, se utiliza una herramienta que se ejecuta en un contenedor llamada portainer, cuenta con una imagen oficial, puede ser ejecutada desde el servidor previamente configurado en Linux con Docker. La figura 3 detalla el comando, que permite la ejecución de un contenedor con la imagen de portainer.

Figura 3.

Comandos para la ejecución de Portainer como contenedor en Docker.

```
docker run -d -p 8000:8000 -p 9443:9443 --name portainer --restart=always \
-v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce:lts
```



Servidor CI/CD

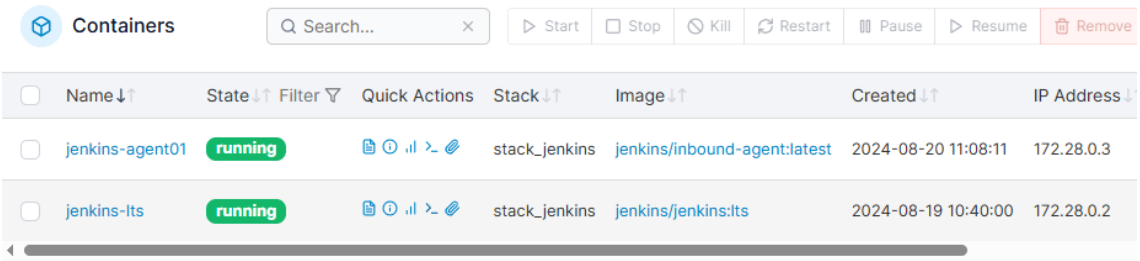
La figura 4 detalla el archivo de configuración Yaml para la ejecución de Jenkins como servidor CI/CD en un entorno de Docker, utilizando un agente para la configuración del pipeline de seguridad DevSecOps.

Figura 4.

Configuración administración contenedores para la ejecución de Jenkins como servidor CI/CD

```yaml
version: '3.7'
services:
  jenkins:
    image: jenkins/jenkins:lts
    privileged: true
    user: root
    ports:
      - 8081:8080
      - 50000:50000
    container_name: jenkins-lts
    volumes:
      - ~/jenkins_home:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
      - /usr/bin/docker:/usr/local/bin/docker
  agent:
    image: jenkins/inbound-agent:latest
    privileged: true
    init: true
    user: root
    container_name: jenkins-agent01

    environment:
      - JENKINS_URL=http://192.168.0.35:8081/
      - JENKINS_AGENT_NAME=agent01
      - JENKINS_SECRET=eb6e5927dff4d22992e92affb5d2dfb4228e4c57e923f8de57b57a259ede1d8e
      - JENKINS_AGENT_WORKDIR=/var/jenkins_home
    volumes:
      - ~/jenkins_home:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
      - /usr/bin/docker:/usr/local/bin/docker
```



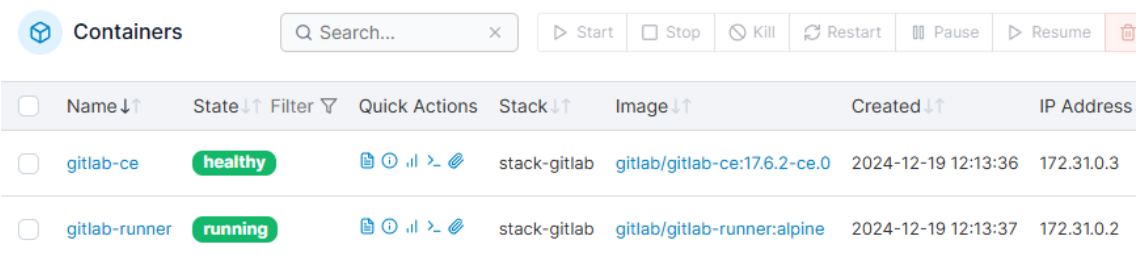
Configuración Servidor Control de Versiones.

En la figura 5 se detalla la configuración Yaml para la ejecución de un Servidor GitLab utilizando contenedores en una arquitectura on-premise.

Figura 5.

Yaml de configuración de servidor Control de Versiones GitLab Opensource para entornos on-premise con Docker.

```yaml
version: '3.7'
services:
  web:
    image: 'gitlab/gitlab-ce:17.6.2-ce.0'
    restart: always
    hostname: 'git.pruebas.com.ec'
    container_name: gitlab-ce
    environment:
      GITLAB_OMNIBUS_CONFIG: |
        external_url 'https://git.pruebas.com.ec'
        nginx['enable'] = true
        nginx['ssl_certificate'] = '/etc/gitlab/ssl/certificado.crt'
        nginx['ssl_certificate_key'] = '/etc/gitlab/ssl/llave.key'
    ports:
      - '80:80'
      - '443:443'
    volumes:
      - '$GITLAB_HOME/config:/etc/gitlab'
      - '$GITLAB_HOME/logs:/var/log/gitlab'
      - '$GITLAB_HOME/data:/var/opt/gitlab'
      - '/srv/gitlab/config/ssl:/etc/gitlab/ssl'
    networks:
      - gitlab
  gitlab-runner:
    image: gitlab/gitlab-runner:alpine
    container_name: gitlab-runner
    restart: always
    depends_on:
      - web
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - '$GITLAB_HOME/gitlab-runner:/etc/gitlab-runner'
    networks:
      - gitlab
networks:
  gitlab:
    name: gitlab-network
```



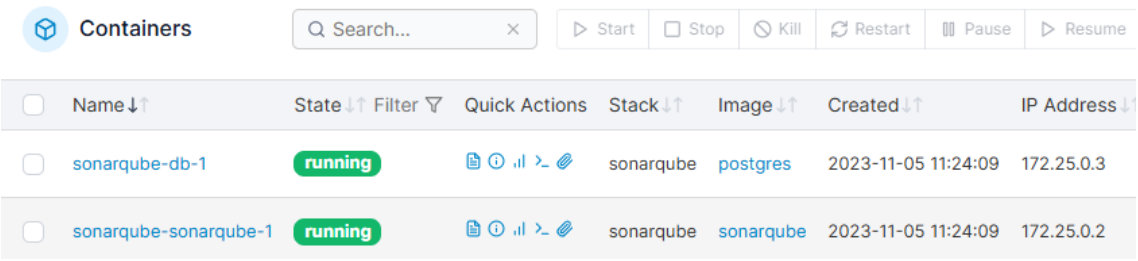
Configuración SonarQube en arquitecturas on-premise

La figura 6 muestra la configuración de como ejecutar la herramienta SonarQube utilizando archivos Yaml compatibles con Docker en arquitecturas locales.

Figura 6.

Configuración de SonarQube en Docker con archivos Yaml.

```yaml
version: "3"
services:
  sonarqube:
    image: sonarqube
    ports:
      - "9000:9000"
    networks:
      - sonarnet
    environment:
      - sonar.jdbc.url=jdbc:postgresql://db:5432/sonar
    volumes:
      - sonarqube_conf:/opt/sonarqube/conf
      - sonarqube_data:/opt/sonarqube/data
      - sonarqube_extensions:/opt/sonarqube/extensions
  db:
    image: postgres
    networks:
      - sonarnet
    environment:
      - POSTGRES_USER=sonar
      - POSTGRES_PASSWORD=Clave12345
    volumes:
      - postgres_db:/var/lib/postgresql/data
networks:
  sonarnet:
    driver: bridge
volumes:
  sonarqube_conf:
  sonarqube_data:
  sonarqube_extensions:
  postgres_db:
```



Herramienta de Gestión de Secretos con Infisical

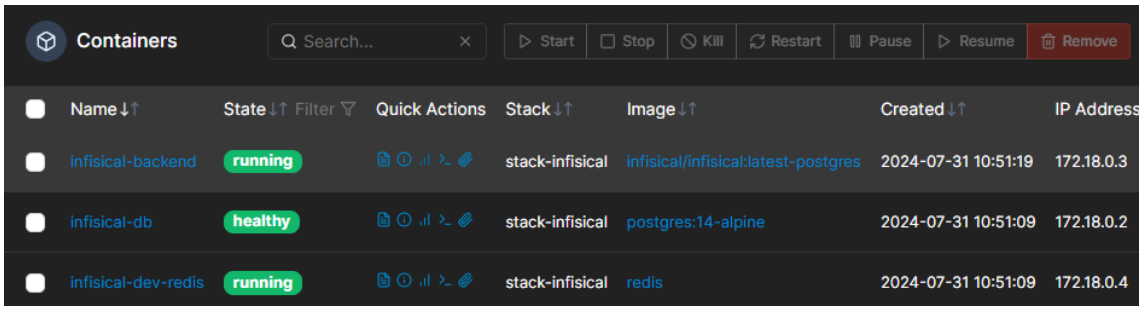
Infisical herramienta de gestión se secretos permite que su ejecución se pueda desplegar en arquitecturas on-premise utilizando contenedores, la figura 7 detalla la configuración utilizando archivos Yaml.

Figura 7.

Configuración de Infisical con Docker para infraestructuras locales.

```yaml
version: "3"
services:
  db-migration:
    container_name: infisical-db-migration
    depends_on:
      db:
        condition: service_healthy
    image: infisical/infisical:latest-postgres
    env_file: stack.env
    command: npm run migration:latest
    pull_policy: always
    networks:
      - infisical
  backend:
    container_name: infisical-backend
    restart: unless-stopped
    depends_on:
      db:
        condition: service_healthy
      redis:
        condition: service_started
      db-migration:
        condition: service_completed_successfully
    image: infisical/infisical:latest-postgres
    pull_policy: always
    env_file: stack.env
    ports:
      - 80:8080
    environment:
      - NODE_ENV=production
    networks:
      - infisical

  redis:
    image: redis
    container_name: infisical-dev-redis
    env_file: stack.env
    environment:
      - ALLOW_EMPTY_PASSWORD=yes
    ports:
      - 6379:6379
    networks:
      - infisical
    volumes:
      - redis_data:/data
  db:
    container_name: infisical-db
    image: postgres:14-alpine
    restart: always
    env_file: stack.env
    volumes:
      - pg_data:/var/lib/postgresql/data
    networks:
      - infisical
    healthcheck:
      test: "pg_isready --username=${POSTGRES_USER} && psql --username=${POSTGRES_USER} --list"
      interval: 5s
      timeout: 10s
      retries: 10
volumes:
  pg_data:
    driver: local
  redis_data:
    driver: local
networks:
  infisical:
```



Microservicio de Prueba

Para la simulación en el entorno controlado de pruebas, se utiliza un microservicio programado con el lenguaje c# con framework Net 8, proporciona un API de autenticación de una aplicación utilizando JWT para la generación de Tokens, la figura 8 muestra la configuración del archivo Dockerfile para la construcción de la imagen de Docker.

Figura 8.

DockerFile Microservicio de Prueba para la compilación y construcción de la imagen de Docker.

```
#STAGE 1
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /app
COPY . ./
RUN dotnet restore WebApplication1.csproj
COPY . ./
RUN dotnet publish WebApplication1.csproj -c release -o out


#STAGE 2
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
WORKDIR /app
COPY --from=build /app/out .
ENV ASPNETCORE_URLS http://*:80
ENV DOTNET_RUNNING_IN_CONTAINER true
EXPOSE 80
ENTRYPOINT ["dotnet", "WebApplication1.dll"]

```

Configuración PipeLine Seguro DevSecOps.

Para la construcción del pipeline se utiliza la herramienta Jenkins, se configura cada fase que se utilizará a lo largo de todo el ciclo de vida del desarrollo de software.

En la figura 9 se muestra las fases para la construcción de un pipeline seguro, utilizando un microservicio generado en lenguaje C# y framework Net 8 de Microsoft. Las fases a configurar son el repositorio de Git desde GitLab instalado de forma local en la infraestructura On-premise, la compilación, el análisis del código fuente, evaluación de dependencias e identificación de vulnerabilidades y despliegue del microservicio.

Figura 9.

Construcción inicial del pipeline seguro en Jenkins.

En la figura 10 se muestra la fase de Git Checkout, donde se configura el repositorio con la versión generada en la rama master del microservicio, Jenkins permite configurar si el proceso se realiza de forma automática, es decir cuando se aplica una unión de ramas merge en el repositorio remoto de GitLab.

Figura 10.

Fase Git con repositorio remoto en GitLab.

Nota. Código de la fase Git para la extracción del código desde un repositorio GitLab.

La figura 11 determina la fase de compilación, Jenkins permite seleccionar la realease del proyecto del microservicio y el framework en el que fue desarrollado. Esta fase permitirá compilar la aplicación y validar errores comunes en la fase de desarrollo utilizando herramientas SAST.

Figura 11.

Fase Compilación.

```
19        stage('Compile SAST') {
20            steps {
21                dotnetBuild configuration: 'Release', project: 'WebApplication1.sln', sdk: '.NET 8'
22            }
23        }
```

Nota. Código para la compilación de la realease del microservicio desarrollado en el framework Net 8 de Microsoft.

SonarQube es configurada en la siguiente fase, la figura 12 muestra la vinculación de la herramienta, la fase permite analizar el código SAST para la detección de vulnerabilidades, problemas de codificación, código duplicado, calidad de código y métricas, ayudando a mejora la calidad y reducir riesgos.

Figura 12.

Fase de Análisis de código utilizando herramienta SonarQube (SAST).

```
stage('Sonar Analyst SCA') {
    agent{
        docker{
            image 'mcr.microsoft.com/dotnet/sdk:8.0'
        }
    }
   steps {
       withSonarQubeEnv('sonarqube') {
              sh 'dotnet ${scannerHome}/SonarScanner.MSBuild.dll begin /k:"entornoprueba-api"  /d:sonar.host.url="http://192.168.0.35:9000/" /d:sonar.login="squ_9e88054fc1ebdf1ebb6bcb0ecbe13e71e2fb1f3f"'
              sh "dotnet build /var/jenkins_home/workspace/CI_ENTORNOPRUEBA/WebApplication1.sln"
              sh 'dotnet ${scannerHome}/SonarScanner.MSBuild.dll end /d:sonar.login="squ_9e88054fc1ebdf1ebb6bcb0ecbe13e71e2fb1f3f"'
       }
   }
}
```

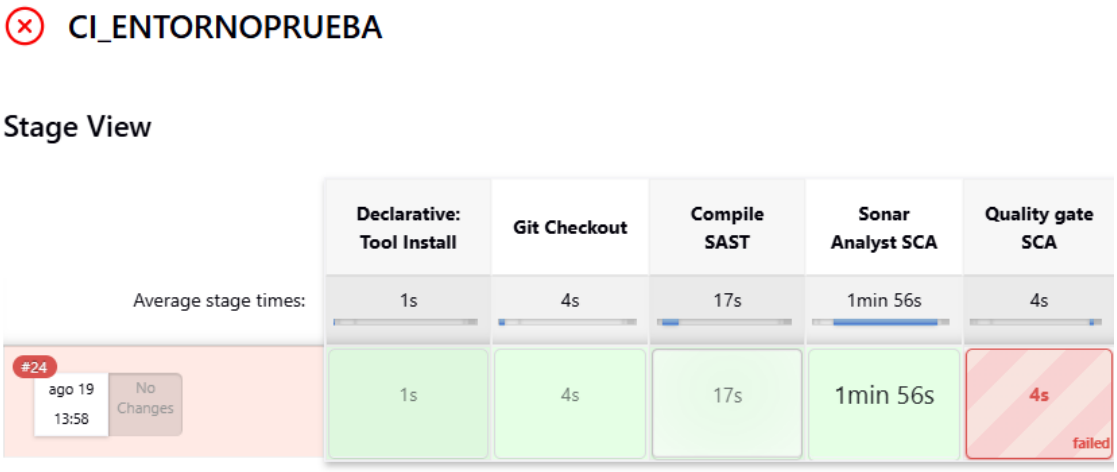
*Nota***.** Código para la configuración de SonarQube utilizando un contenedor, analiza y detecta vulnerabilidades de seguridad.

Para identificar los resultados de la herramienta SAST se utiliza una fase de validación llamada Quality Gaste (SCA), donde se muestra en la figura 13, la configuración en el pipeline. Si SonarQube responde detecta que existen vulnerabilidades o problemas en el código, no permite la continuidad a la siguiente fase.

Figura 13.

Fase de validación de calidad de código de la herramienta SonarQube.

```
40        stage("Quality gate SCA") {
41            steps {
42                waitForQualityGate abortPipeline: true
43            }
44        }
```



Nota. Código de la fase de Validación del análisis de resultado de SonarQube.

Para la construcción del contenedor se muestra la configuración de la siguiente fase en la figura 14, una vez validado el código por SonarQube se genera la construcción de la imagen de Docker antes de su despliegue a producción.

Figura 14.

Fase de construcción de contenedor.

```
46        stage('Docker Build DAST') {
47            steps {
48               dir('WebApplication1') {
49                    sh "docker build -t ${IMAGE_NAME}:${TAG} ."
50               }
51            }
52        }
```

Nota. Código de la base para la construcción de la imagen del contenedor.

La fase para DAST, se muestra en la figura 15, se utiliza la herramienta OWASP ZAP, para analizar vulnerabilidades de seguridad en aplicaciones web que se ejecutan en el contenedor, en este ejemplo el API del microservicio.

Figura 15.

Fase DAST, análisis de vulnerabilidades para API en microservicios.

```
stage('DAST - ZAP') {
   environment {
     APP_IMAGE = 'stalin9116/entornoprueba-api'
     APP_PORT  = '8080'
     HOST_PORT = '8080'
     APP_NAME  = 'app-test'
     TARGET    = "http://localhost:${HOST_PORT}"
 }
 steps {
   sh '''
     docker rm -f $APP_NAME || true
     docker run -d --name $APP_NAME -p $HOST_PORT:$APP_PORT $APP_IMAGE
     for i in {1..40}; do curl -fsS $TARGET/health && break || sleep 3; done
     mkdir -p zap
     docker run --rm --network host -v $PWD/zap:/zap/wrk:rw owasp/zap2docker-stable zap-baseline.py -t $TARGET -r report.html -x report.xml
     [ $(grep -c '<riskcode>3<' zap/report.xml || true) -eq 0 ]
   '''
 }
 post {
   always {
     archiveArtifacts artifacts: 'zap/*', fingerprint: true
     sh 'docker rm -f $APP_NAME || true'
   }
 }
}
```

Nota. Código para implementar la fase de DAST utilizando Owasp Zap.

Continuando con la siguiente fase, se vincula la herramienta Trivy Scan (DAST) que se muestra en la figura 16, permite el análisis de seguridad del contenedor compilado, también errores de configuración en la imagen, vulnerabilidades, sistema de archivos, sistema operativo.

Figura 16.

Fase DAST herramienta Trivy Scan, análisis de vulnerabilidades.

```
54    stage('Trivy Scan DAST') {
55    steps {
56        script {
57            def result = sh(
58                script: """
59                    docker run --rm \
60                        -v /var/run/docker.sock:/var/run/docker.sock \
61                        aquasec/trivy:latest image \
62                        --severity CRITICAL,HIGH \
63                        ${IMAGE_NAME}:${TAG}
64                """,
65                returnStatus: true
66            )
67
68            if (result != 0) {
69                error "Trivy encontró vulnerabilidades CRÍTICAS o ALTAS. Deteniendo el pipeline."
70            }
71        }
72    }
73 }
```

Nota. Código del pipeline permite agregar la herramienta de Trivy Scan para análisis de vulnerabilidades en contenedores.

Para la publicación de la imagen en repositorios remotos de Docker, se muestra en la figura 17 la configuración de la fase utilizando como referencia Docker Hub. Las credenciales pueden configurarse en una base de datos privada de Jenkins para no exponer credenciales en las distintas configuraciones de cada fase.

Figura 17.

Fase de publicación de imágenes en repositorios remotos. DockerHub.

```
76        stage('Docker Push SAST') {
77            steps {
78                withCredentials([usernamePassword(credentialsId: 'dockerhub-creds', usernameVariable: 'DOCKER_USER', passwordVariable: 'DOCKER_PASS')]) {
79                    sh """
80                        echo "$DOCKER_PASS" | docker login -u "$DOCKER_USER" --password-stdin
81                        docker push ${IMAGE_NAME}:${TAG}
82                        docker logout
83                    """
84                }
85            }
86        }
```

Nota. Código de la fase de publicación utilizando el repositorio remoto DockerHub.

2.3 Pasos aplicación Etapa 3 - Implementación e Integración Técnica

Para la implementación e integración del modelo propuesto se indica en la tabla 14 los pasos, instrumentos y los resultados esperados en cada proceso, para llevar una implementación controlada.

Tabla 14.

Pasos etapa 3 para la implementación del modelo DevSecOps.

No.	Proceso	Pasos de Implementación	Instrumentos	Resultado Esperado
1	Despliegue en Infraestructura On-Premise	Instalar y configurar servidores locales, contenedores y repositorios privados de imágenes.	Servidores locales, Docker, registros privados	Infraestructura On-Premise preparada para soportar el pipeline DevSecOps.
2	Integración de Herramientas en el Pipeline	Conectar Jenkins con GitLab, SonarQube, OWASP ZAP, Dependency-Check, Trivy, Vault, etc.	Jenkinsfile, plugins CI/CD, conectores de seguridad.	Pipeline con integración completa de herramientas de automatización y seguridad.
3	Automatización de Escaneos de Seguridad	Incluir análisis estático (SAST), dinámico (DAST), dependencias (SCA) y secretos en el flujo.	SonarQube, OWASP ZAP, Dependency-Check, Trivy, Infisical/Vault.	Ejecución automática de controles de seguridad en cada commit o despliegue.

4	Configuración de Alertas y Bloqueo	Definir quality gates y reglas de bloqueo según criticidad de vulnerabilidades detectadas.	SonarQube Quality Gates, Jenkins Pipelines, sistemas de notificación (Slack/Email).	Pipeline capaz de detener despliegues ante vulnerabilidades críticas.
5	Validación Operativa del Pipeline	Ejecutar pruebas en entornos controlados simulando despliegues reales.	Jenkins logs, reportes de escaneo, contenedores de prueba.	Pipeline validado funcionalmente con resultados reproducibles.
6	Monitoreo Inicial y Ajustes	Revisar métricas de ejecución, consumo de recursos y tiempo de respuesta. Ajustar parámetros.	Prometheus, Grafana, métricas de Jenkins.	Pipeline optimizado en rendimiento y confiabilidad, listo para su uso regular.

2.4 Pasos aplicación Etapa 4 - Verificación y Mejora Continua

En la fase final, la tabla 15 detalla los pasos, instrumentos y resultados esperados en cada proceso del modelo propuesto DevSecOps.

Tabla 15.

Pasos fase 4 final Verificación y mejora continua.

No.	Proceso	Pasos de Implementación	Instrumentos	Resultado Esperado
1	Verificación del Modelo en Entorno Controlado	Ejecutar el pipeline DevSecOps en un entorno de simulación On-Premise. Validar la integración completa de las herramientas.	Entorno de pruebas, Jenkins logs, reportes de escaneo.	Modelo verificado en condiciones controladas.
2	Evaluación de Detección y Respuesta	Medir el tiempo de identificación de vulnerabilidades y la efectividad de los mecanismos de respuesta automática.	Métricas de SonarQube, OWASP ZAP, Dependency-Check, Prometheus/Grafana.	Evidencia cuantitativa del nivel de detección y velocidad de respuesta.
3	Análisis de Datos Generados	Revisar logs, métricas y reportes generados durante la ejecución del pipeline. Identificar patrones y áreas de mejora.	Dashboards de Grafana, registros de auditoría, bases de datos de vulnerabilidades.	Informe de análisis técnico con indicadores clave de seguridad y eficiencia.

4	Retroalimentación de Equipos Técnicos	Realizar reuniones de revisión con equipos Dev, Ops y Seguridad para discutir hallazgos y sugerencias.	Actas de reunión, focus groups técnicos, cuestionarios de retroalimentación.	Conjunto de recomendaciones prácticas para ajustar el modelo.
5	Ajustes de Configuración y Refinamiento	Implementar mejoras en políticas de calidad, reglas de bloqueo y configuraciones de herramientas.	Jenkins pipeline actualizado, configuración de SonarQube y OWASP ZAP ajustada.	Pipeline optimizado con base en hallazgos empíricos y feedback del equipo.
6	Establecimiento de Ciclo de Mejora Continua	Aplicar la metodología PDCA (Plan-Do-Check-Act) para asegurar un ciclo constante de optimización.	Ciclos PDCA documentados, métricas de evolución continua.	Modelo DevSecOps evolucionando de manera sistemática con base en métricas y lecciones.

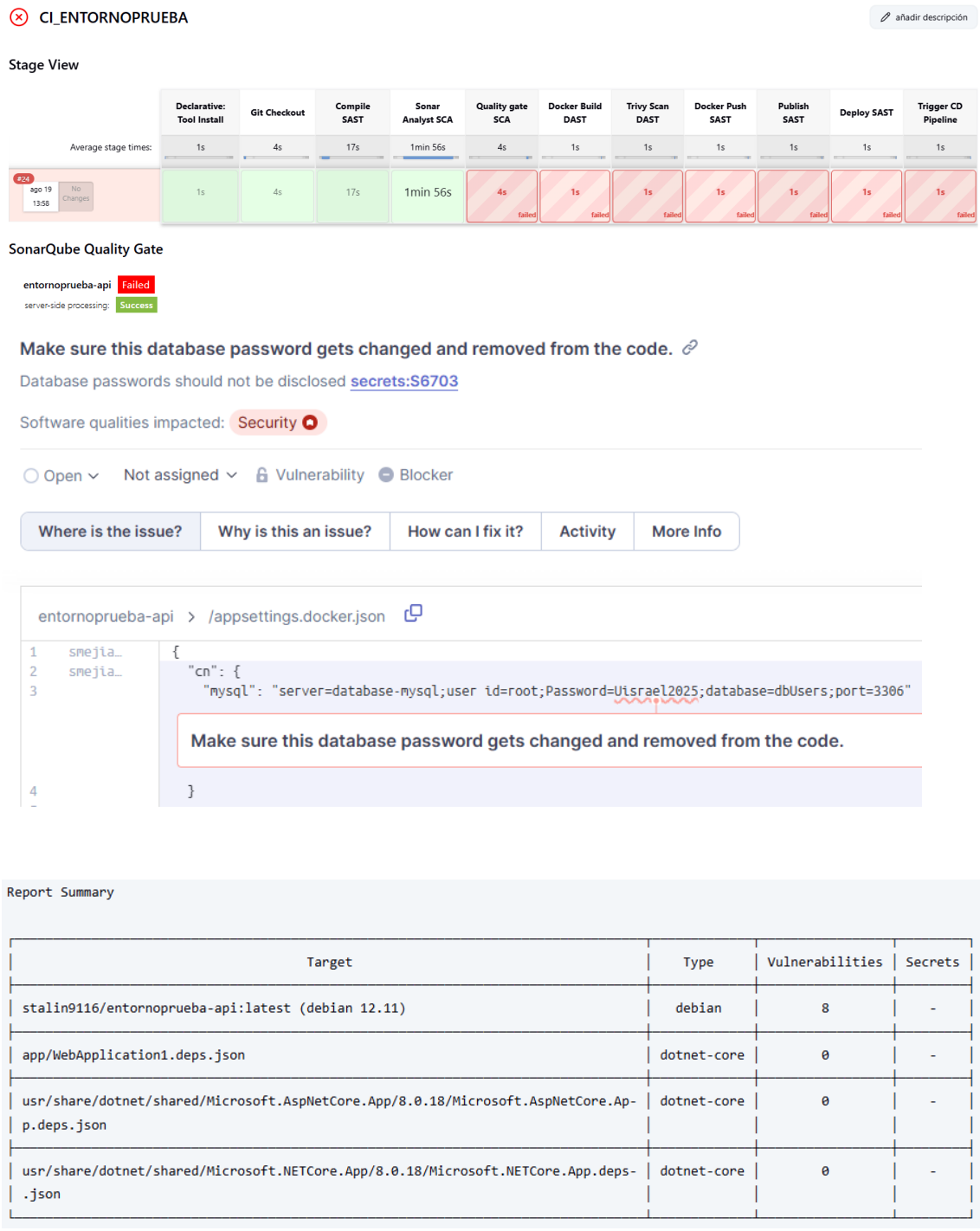
3. Evaluación y Validación del Modelo

La valoración del esquema propuesto se apoya en la ejecución del pipeline seguro diseñado para instalaciones locales. Los criterios de análisis abarcan tres frentes: la integración efectiva de controles automáticos en cada fase del ciclo de vida, la capacidad del flujo para detectar debilidades en etapas tempranas y la resiliencia de la infraestructura CI/CD ante escenarios simulados. Como indicadores de efectividad se miden la reducción de vulnerabilidades críticas en producción, el tiempo medio entre hallazgo y corrección, la tasa de falsos positivos y la continuidad de los despliegues sin pausas innecesarias. En cuanto a los resultados esperados, se proyecta un progreso notable en la madurez de seguridad de la organización, visible en métricas como una disminución cercana al 50 % de fallos graves, recortes significativos en los tiempos de mitigación y una mayor fiabilidad de los lanzamientos. Estos datos permitirán confirmar que la propuesta genera valor técnico y cultural, al fortalecer la disciplina de seguridad y la sostenibilidad del desarrollo sobre arquitecturas de microservicios.

Para la fase de evaluación y validación del Modelo DevSecOps se realizan pruebas de ejecución del pipeline seguro que se generó en la fase 2 de construcción, a continuación, se muestra en la figura 18 las fases vinculadas al modelo DevSecOps, la evaluación y resultados que permiten controlar las vulnerabilidades encontradas en el microservicio de prueba.

Figura 18.

Evaluación del modelo DevSecOps en Jenkins.




```
Report Summary

┌──────────────────────────────────────────────────────────────────────────────┬────────────┬─────────────────┬─────────┐
│                                    Target                                    │    Type    │ Vulnerabilities │ Secrets │
├──────────────────────────────────────────────────────────────────────────────┼────────────┼─────────────────┼─────────┤
│ stalin9116/entornoprueba-api:latest (debian 12.11)                           │   debian   │        8        │    -    │
├──────────────────────────────────────────────────────────────────────────────┼────────────┼─────────────────┼─────────┤
│ app/WebApplication1.deps.json                                                │ dotnet-core │        0        │    -    │
├──────────────────────────────────────────────────────────────────────────────┼────────────┼─────────────────┼─────────┤
│ usr/share/dotnet/shared/Microsoft.AspNetCore.App/8.0.18/Microsoft.AspNetCore.Ap- │ dotnet-core │        0        │    -    │
│ p.deps.json                                                                  │            │                 │         │
├──────────────────────────────────────────────────────────────────────────────┼────────────┼─────────────────┼─────────┤
│ usr/share/dotnet/shared/Microsoft.NETCore.App/8.0.18/Microsoft.NETCore.App.deps- │ dotnet-core │        0        │    -    │
│ .json                                                                        │            │                 │         │
└──────────────────────────────────────────────────────────────────────────────┴────────────┴─────────────────┴─────────┘
```

Los resultados de la ejecución del pipeline identifican vulnerabilidades en las diferentes fases de integración y despliegue continuo CI/CD, ya que las herramientas que se utilizaron (SAST, DAST, SCA) al analizar el código del microservicio muestran las advertencias de cada vulnerabilidad encontrada, evitando el paso a producción del microservicio y la construcción de la imagen en Docker.

4. Aplicaciones y Beneficios del Modelo

El esquema DevSecOps propuesto ofrece aplicaciones inmediatas y beneficios concretos para organizaciones que gestionan microservicios sobre infraestructura local. En primer lugar, la solución automatiza la detección y la mitigación de fallos dentro de los pipelines de integración y entrega continua, lo que disminuye de forma notable la probabilidad de que vulnerabilidades críticas alcancen el entorno de producción.

En el plano cultural, el modelo impulsa la cooperación entre desarrollo, operaciones y seguridad, desmonta silos tradicionales y reparte la responsabilidad de protección a lo largo de todo el ciclo de vida del software. El empleo de herramientas de código abierto, además, mantiene bajo control los costes de licencia y facilita la adaptación a diferentes contextos corporativos.

Un beneficio adicional reside en la reducción de tiempos y gastos asociados a la corrección de errores: solventar una vulnerabilidad en fases iniciales del SDLC puede resultar hasta quince veces más económico que hacerlo tras el despliegue. Desde la perspectiva estratégica, la adopción del modelo favorece el alineamiento con marcos normativos reconocidos como ISO 27001, NIST, OWASP, aumentando la confianza de clientes y socios.

Trabajo Final

INFORME DE ORIGINALIDAD

9%	9%	3%	4%
INDICE DE SIMILITUD	FUENTES DE INTERNET	PUBLICACIONES	TRABAJOS DEL ESTUDIANTE

FUENTES PRIMARIAS

1	www.ismsforum.es Fuente de Internet	1%
2	republicanaradio.com Fuente de Internet	1%
3	devblogs.microsoft.com Fuente de Internet	1%
4	repository.unad.edu.co Fuente de Internet	1%
5	www.dspace.uce.edu.ec Fuente de Internet	1%
6	repositorio.uladech.edu.pe Fuente de Internet	1%
7	www.coursehero.com Fuente de Internet	1%
8	repositorio.utn.edu.ec Fuente de Internet	1%
9	repository.libertadores.edu.co Fuente de Internet	1%

*% detectado como IA

La detección de IA incluye la posibilidad de que haya falsos positivos. Aunque cierto texto en esta entrega se generó probablemente con IA, los puntajes inferiores al umbral del 20 % no aparecen porque tienen una mayor probabilidad de falsos positivos.

Precaución: Se necesita revisión.

Es esencial comprender los límites de la detección de IA antes de tomar decisiones acerca del trabajo del estudiante. Te alentamos a obtener más información acerca de las funciones de detección de IA de Turnitin antes de usar la herramienta.