UNIVERSIDAD MAYOR, REAL Y PONTIFICIA DE SAN FRANCISCO XAVIER DE CHUQUISACA

VICERRECTORADO

CENTRO DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

FACULTAD DE CIENCIAS Y TECNOLOGÍA

"EXPLORACIÓN DE HERRAMIENTAS DE ANÁLISIS ESTÁTICO PARA SCRIPTS DE IaC: UNA REVISIÓN SISTEMÁTICA"

TRABAJO EN OPCIÓN A DIPLOMADO EN DEVELOPMENT OPERATIONS "DEVOPS" V.1.

AUTOR: ANDRES COLQUE ROSAS

SUCRE-BOLIVIA

2024

CESIÓN DE DERECHOS

Al presentar este trabajo como requisito previo para la obtención del Diploma en Development Operations "DEVOPS" V.1. de la Universidad Mayor, Real y Pontificia de San Francisco Xavier de Chuquisaca, autorizo al Centro de Estudios de Posgrado e Investigación o a la Biblioteca de la Universidad, para que se haga de este trabajo un documento disponible para su lectura según normas de la Universidad

También cedo a la Universidad Mayor, Real y Pontifica de San Francisco Xavier de Chuquisaca, los derechos de publicación de este trabajo o parte de él, manteniendo mis derechos de autor hasta un periodo de 30 meses posterior a su aprobación.

Andres Colque Rosas

……..

FIRMA

RESUMEN

La Infraestructura como Código (IaC) automatiza la gestión de infraestructuras, mejorando la eficiencia y consistencia en su administración. Sin embargo, la complejidad y diversidad de las tecnologías utilizadas presentan desafíos en la detección de errores y vulnerabilidades en los scripts de IaC. Este estudio evalúa herramientas de análisis estático para mejorar la calidad y seguridad de estas infraestructuras, proporcionando una guía para su selección adecuada.

El objetivo del estudio es evaluar y clasificar las herramientas de análisis estático para scripts de IaC según sus características técnicas y operacionales, facilitando su selección en entornos DevOps. Se realiza una revisión sistemática de la literatura y un análisis comparativo de 21 herramientas utilizando métodos documentales y bibliográficos.

Se identifican 21 herramientas, destacando Checkov, KICS, Snyk IaC, Terrascan, Tfsec y Trivy por su capacidad para detectar errores y vulnerabilidades. Algunas herramientas sobresalen por su integración con CI/CD y soporte para múltiples lenguajes, lo que las hace más versátiles y útiles en diversos entornos de desarrollo. Además, se presenta una guía de preguntas detalladas para ayudar en la selección de la herramienta más adecuada según el entorno y los requerimientos específicos.

La selección de herramientas considera tanto las características técnicas como las operacionales. Es esencial realizar pruebas prácticas, capacitar continuamente a los equipos y desarrollar nuevas técnicas de detección utilizando inteligencia artificial, combinadas con otras prácticas de seguridad.

Este estudio proporciona una evaluación exhaustiva y una clasificación de herramientas de análisis estático para scripts de IaC, además de una guía práctica de preguntas para una correcta selección. Estas contribuciones mejoran la toma de decisiones en la implementación de IaC, optimizando la calidad y seguridad de las infraestructuras gestionadas mediante este enfoque.

INDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

1 Antecedentes y Justificación

Antecedentes

La Infraestructura como Código (IaC) revolucionó la gestión de la configuración de software, automatizando el desarrollo y el despliegue de sistemas. Este enfoque mejoro significativamente la eficiencia y consistencia en la administración de infraestructuras complejas. Sin embargo, la creciente complejidad de los entornos de infraestructura y la diversidad de tecnologías utilizadas generaron desafíos importantes en la detección y corrección de posibles vulnerabilidades, errores de configuración y *code smells* en los *scripts* de IaC (Morris, 2016).

Un estudio realizado por (Jiang y Adams, 2015), se observó que los scripts de Infraestructura como Código suelen ser modificados por desarrolladores regulares en lugar de expertos en infraestructura, lo que puede resultar en una baja calidad del código y aumentar el riesgo de errores de configuración de seguridad. Por lo tanto, se resaltó la necesidad de implementar métodos automatizados para detectar estos riesgos, lo que llevó a la recomendación de investigar más sobre herramientas y técnicas para abordar esta problemática.

En otra investigación llevada a cabo por (Sharma et al., 2016), se realizó un análisis preliminar de la calidad del código de configuración examinando 4.621 repositorios Puppet que contenían 142.662 archivos Puppet y más de 8,9 millones de líneas de código en busca de problemas de configuración. Se descubrió que los *security smells* pertenecientes a una categoría específica a menudo coexisten con otros *security smells* de diferentes categorías en el código de configuración.

Además, en la investigación realizada por (Guerriero et al., 2019) llevaron a cabo 44 entrevistas con profesionales de Tecnologías de la Información (TI) para recopilar información sobre sus prácticas en el desarrollo de Infraestructura como Código (IaC) y el software que utilizan comúnmente. Los resultados indican que Ansible es el software más empleado entre los encuestados, junto con una variedad de herramientas como Docker, Kubernetes, Vagrant, Chef, Terraform, entre otros, que también son ampliamente utilizados en el ámbito de IaC. Se destacó la necesidad de comprender las demandas de los profesionales de TI en términos de desarrollo, mantenimiento y evolución de IaC, así como la tendencia a utilizar una combinación de herramientas especializadas debido a la falta de una solución integral que abarque todos los aspectos de IaC.

En un estudio más reciente realizado por (Chiari et al., 2022), se llevó a cabo una investigación exhaustiva de herramientas de análisis de configuración estática. Sin embargo, el estudio se centró principalmente en herramientas que cuentan con respaldo académico, lo cual podría dejar de lado técnicas prácticas que no están formalmente documentadas. Esta limitación destaca la necesidad de incluir una variedad más amplia de herramientas, incluidas aquellas que no están ampliamente cubiertas en la literatura académica, para obtener una comprensión completa de las opciones disponibles y sus capacidades.

Justificación

La adopción de la Infraestructura como Código (IaC) revolucionó la gestión y automatización de infraestructuras de TI, permitiendo a los equipos definir, desplegar y gestionar infraestructuras a través de código. Sin embargo, este avance también introduce desafíos significativos relacionados con la calidad, seguridad y mantenimiento del código. Los errores y vulnerabilidades en los scripts de IaC pueden tener consecuencias graves, como fallos en el despliegue, brechas de seguridad y tiempos de inactividad. Por lo tanto, es esencial contar con herramientas de análisis estático que detecten problemas potenciales antes de que el código sea desplegado.

La justificación de esta investigación radica en la necesidad de evaluar de manera integral y sistemática las herramientas de análisis estático disponibles para scripts de IaC. Actualmente, existen numerosas herramientas en el mercado, cada una con diferentes enfoques, técnicas y metodologías para la detección de errores y vulnerabilidades. Sin una guía clara y basada en evidencia, los profesionales de DevOps y las organizaciones pueden enfrentar dificultades para seleccionar las herramientas más adecuadas para sus necesidades específicas. Esta investigación busca proporcionar esa guía, facilitando una selección informada y eficiente de herramientas de análisis estático en los scripts de IaC, así contribuir a la mejora de la calidad y seguridad de las infraestructuras gestionadas mediante este enfoque.

2 Situación Problemática

La gestión de infraestructura de TI a través de scripts automatizados, conocida como Infraestructura como Código (IaC), ha transformado la manera en que las organizaciones desarrollan y despliegan sus sistemas. Este enfoque permite a los equipos definir y gestionar su infraestructura utilizando código, lo que aporta beneficios significativos en términos de eficiencia, repetibilidad y escalabilidad. No obstante, esta misma práctica introduce una serie

de desafíos críticos, particularmente en relación con la calidad y seguridad del código empleado.

Una de las principales problemáticas radica en la detección y corrección de vulnerabilidades y errores de configuración en los scripts de IaC. Dada la complejidad creciente de los entornos de infraestructura y la variedad de tecnologías empleadas, identificar estos problemas de manera oportuna se vuelve cada vez más difícil. Los errores y vulnerabilidades no detectados pueden llevar a fallos graves en el despliegue, brechas de seguridad y otros incidentes que pueden afectar la estabilidad y seguridad de la infraestructura de TI.

Actualmente, muchas organizaciones dependen de métodos manuales o herramientas limitadas para la revisión de scripts de IaC, lo cual resulta insuficiente frente a la escala y complejidad de las infraestructuras modernas. Estos métodos manuales no solo son propensos a errores humanos, sino que también resultan insostenibles a medida que los proyectos aumentan en tamaño. La falta de herramientas automatizadas eficaces que realicen análisis estáticos de manera integral y precisa representa un riesgo significativo, ya que los problemas de configuración y *security smells* pueden pasar desapercibidos o no ser gestionados adecuadamente.

Además, hay una carencia notable de estudios que aborden de manera exhaustiva las herramientas de análisis estático para scripts de IaC. Las investigaciones existentes a menudo se centran en herramientas específicas o en casos de uso particulares, dejando de lado una evaluación amplia que considere diversas técnicas y metodologías. Esta brecha en el conocimiento impide que los profesionales de DevOps y las organizaciones tengan una visión completa y comparativa de las opciones disponibles, lo que dificulta la selección de las herramientas más adecuadas para sus necesidades específicas.

3 Formulación del Problema de Investigación.

¿Cuáles son las herramientas y técnicas más efectivas para el análisis estático para scripts de IaC y cómo pueden clasificarse según sus características técnicas y operacionales para facilitar su selección en entornos DevOps?

4 Objetivo General

Evaluar el estado actual de las herramientas de análisis estático para scripts de Infraestructura como Código (IaC) mediante un análisis detallado de la literatura disponible, con el fin de facilitar su correcta selección y uso en entornos DevOps.

5 Objetivos Específicos

- Identificar las herramientas de análisis estático más utilizadas en scripts de Infraestructura como Código (IaC).
- Clasificar las herramientas de análisis estático para scripts de Infraestructura como Código (IaC) según sus características técnicas.
- Clasificar las herramientas de análisis estático para scripts de Infraestructura como Código (IaC) según sus características operacionales.
- Elaborar un conjunto de preguntas que guíen la selección eficiente de herramientas de análisis estático, adaptadas a scripts de IaC.

6 Diseño Metodológico

6.1 Tipo de Investigación

La investigación propuesta se enfocó en analizar de manera teórico-descriptiva y analítico-descriptiva las herramientas y técnicas utilizadas para el análisis de configuración estática en scripts de Infraestructura como Código (IaC). El objetivo principal fue examinar en detalle las características, funcionalidades y limitaciones de las herramientas de análisis estático. Para lograrlo, se realizó una revisión sistemática de la literatura y un análisis comparativo a través de métodos de investigación documental y bibliográfica. Este enfoque permitió identificar patrones, tendencias y áreas de mejora específicas en el análisis estático de IaC.

6.1.1 Alcance de la Investigación

La investigación se centró en una revisión sistemática y comparativa de las herramientas de análisis estático disponibles para scripts de Infraestructura como Código (IaC). El objetivo principal fue evaluar estas herramientas en términos de características, funcionalidades, técnicas de detección y metodologías, proporcionando una evaluación crítica y exhaustiva que permita una correcta selección y uso en entornos DevOps.

En este sentido, el alcance abarco:

- Identificación de herramientas de análisis estático: Se identificaron las herramientas de análisis estático más utilizadas en scripts de IaC, tanto en el ámbito académico como en la industria. Esta identificación permitió establecer una base amplia de herramientas para su posterior análisis.

- Clasificación de herramientas según características técnicas: Las herramientas identificadas se clasificaron según sus características técnicas, tales como los lenguajes y plataformas soportadas, técnicas de detección utilizadas, y capacidades de integración con sistemas de control de versiones y pipelines de CI/CD.
- Clasificación de herramientas según características operacionales: Además de las características técnicas, las herramientas se clasificaron según sus características operacionales, incluyendo facilidad de uso, soporte y documentación, licencia o costo, y su adopción en entornos reales de DevOps.
- Análisis comparativo de herramientas: Se realizó un análisis comparativo de las herramientas para evaluar sus fortalezas y debilidades. Este análisis incluyó la evaluación de su efectividad en la detección de errores de configuración, seguridad y conformidad, así como en la mejora de la mantenibilidad y legibilidad del código.
- Elaboración de preguntas guía para la selección de herramientas: Se desarrolló un conjunto detallado de preguntas que guíen la eficiente selección de herramientas de análisis estático, adaptadas a scripts de IaC. Estas preguntas fueron diseñadas para ayudar a los profesionales a tomar decisiones informadas basadas en las necesidades específicas de sus proyectos y entornos de trabajo.

Este estudio se centra exclusivamente en el análisis documental, sin abordar la implementación de correcciones ni la realización de pruebas prácticas de las herramientas de análisis de código estático de IaC. Estas áreas se proponen como posibles temas para investigaciones futuras.

6.2 Métodos

En esta investigación se utilizaron los siguientes métodos:

6.2.1 Método Teóricos

Revisión sistemática: Se recopiló y analizó literatura académica y herramientas disponibles en el mercado para identificar técnicas reconocidas en el análisis estático de scripts de IaC.

Análisis Documental y de Contenido: Se exploraron y codificaron textos de documentos seleccionados para identificar patrones y tendencias relevantes.

6.2.2 Método Análisis - Síntesis

Meta-análisis: Se llevó a cabo un meta-análisis de los estudios y fuentes revisadas para sintetizar la información recopilada, identificar tendencias y brechas en el análisis estático de scripts de IaC. Este enfoque permite consolidar los conocimientos adquiridos, identificar las herramientas más efectivas y menos efectivas, y proporcionar una visión general que ayude a profesionales y organizaciones a seleccionar las herramientas adecuadas para sus necesidades. Además, se identificará áreas donde se requiere más investigación y desarrollo para abordar las limitaciones actuales en el análisis estático de IaC.

6.3 Técnica

Para la recolección y análisis de datos en esta investigación sobre análisis estático para scripts de Infraestructura como Código, se emplearon técnicas meticulosamente seleccionadas que apoyaron el entendimiento teórico y la síntesis de la información recopilada. Estas técnicas fueron fundamentales para abordar los objetivos de estudio desde una perspectiva integral y crítica.

- Análisis documental: Esta técnica fue esencial para la exploración exhaustiva de literatura académica y documentos técnicos relevantes. Facilitó la identificación y recolección de datos primarios y secundarios pertinentes al tema de análisis estático. El análisis documental permitió un enfoque sistemático para desglosar y entender los desarrollos y contribuciones en el campo, asegurando que todos los aspectos relevantes de la literatura existente fueran considerados y evaluados.
- Análisis de contenido: El análisis de contenido se aplicó para examinar y codificar de manera estructurada el texto y contenido de los documentos seleccionados. Esta técnica fue crucial para identificar patrones, temas recurrentes y tendencias dentro del ámbito del análisis estático. Permitió la clasificación y cuantificación de información, lo que contribuyó significativamente a la construcción de un marco teórico robusto y a la fundamentación de las discusiones y conclusiones de la investigación.

6.4 Procedimientos e Instrumentos de Investigación

6.4.1 Procedimientos

- Selección de fuentes: Se identificaron y seleccionaron fuentes académicas y técnicas pertinentes a través de bases de datos especializadas como *IEEE Xplore*, *ACM Digital*

Library, y *Google Scholar*. Se priorizaron estudios recientes que ofrecieron información sobre el estado actual y las innovaciones en el análisis estático de IaC.

- Recolección de datos: Los datos fueron recopilados exclusivamente de fuentes documentales:
  - Fuentes primarias: Incluyeron artículos de revistas científicas y conferencias, tesis y disertaciones que proporcionan datos originales y análisis directos sobre herramientas de análisis estático.
  - Fuentes secundarias: Comprenden libros, capítulos de libros y artículos de revisión que resumen y discuten los hallazgos de las fuentes primarias, proporcionando un contexto más amplio y comparaciones entre diferentes estudios y herramientas.
- Evaluación de la información: Se evalúa la validez y la fiabilidad de la información obtenida, considerando la reputación de las fuentes, la coherencia de los argumentos presentados, y la relevancia con respecto a los objetivos de investigación.
- Síntesis de datos: Se organizó la información por temas para crear un marco completo que relacionara los descubrimientos con los objetivos de la investigación.

6.4.2 Instrumentos de Investigación

- Software de gestión bibliográfica: Se utilizaron para organizar las referencias y documentos recopilados, facilitando el acceso rápido durante la redacción y asegurando la precisión en la citación de fuentes.
- Microsoft Excel: Se empleará para crear bases de datos que permitan organizar y analizar cuantitativamente la información sobre las publicaciones, incluyendo autores, año de publicación, temas tratados, y metodologías utilizadas. Esto facilita la identificación de tendencias y patrones.
- Software de análisis de contenido: Se utilizo para realizar análisis cualitativo de textos largos, permitiendo codificar y categorizar el contenido textual de manera sistemática y explorar relaciones complejas entre los datos.
- Programas de procesamiento de texto (Microsoft Word, Google Docs): Se usaron para la redacción del documento final, permitiendo la edición, formateo y revisión del texto conforme a las normas académicas requeridas.

CAPÍTULO I

1 MARCO TEÓRICO Y CONTEXTUAL

1.1 Marco Teórico

1.1.1 Infraestructura como Código (IaC)

La Infraestructura como Código es una práctica innovadora en la gestión de infraestructuras de TI, que consiste en administrar y provisionar dichos recursos a través de archivos de código, en lugar de realizar procesos manuales o utilizar herramientas de configuración ad hoc. Esta metodología permite a los equipos de TI automatizar la configuración de la infraestructura, lo que hace posible la implementación rápida y eficiente de entornos. Además, facilita la consistencia y la estandarización de los entornos, minimizando el riesgo de errores humanos y aumentando la eficacia en la gestión de los recursos tecnológicos (Morris, 2016)

1.1.2 Categorías de Infraestructuras como Código

La adopción de los principios de DevOps ha impulsado la evolución de la gestión de la configuración, pasando de scripts de *shell* a la Infraestructura como Código (IaC). Este enfoque automatiza el aprovisionamiento de infraestructura de TI mediante código, reemplazando la configuración manual.

Para una gestión efectiva de la infraestructura, los conceptos de IaC se dividen en tres categorías distintas, cada una con su propio conjunto de software especializado. Estas categorías son:

- Aprovisionamiento de infraestructura: Se encarga de la automatización del aprovisionamiento y la gestión de servidores, almacenamiento y dispositivos de red tanto físicos como virtuales. Este proceso abarca desde la configuración física de los servidores hasta la instalación del sistema operativo y la configuración de la red. Ejemplos de software especializado en esta área incluyen Terraform, TOSCA y Cloudify.
- Gestión de configuración: Se encarga de la automatización de la configuración y gestión de software y servicios en servidores. Este proceso abarca la instalación de software, la configuración de ajustes y la aplicación de parches. Ejemplos de software en esta categoría incluyen Ansible, Chef y Puppet.
- Creación de imágenes: Se dedica a generar imágenes de máquinas que facilitan la rápida creación de nuevas instancias de servidores. Este proceso implica la creación

de una imagen base de un sistema operativo y su posterior captura como plantilla para la generación de nuevos servidores o instancias. Ejemplos de software utilizado para la creación de imágenes son HashiCorp Packer y Docker (Wang, 2022).

1.1.3 Análisis estático

El análisis estático es una técnica crucial en la revisión del código fuente, que permite inspeccionar el código sin necesidad de ejecutarlo. Esta metodología resulta ser de gran utilidad para identificar problemas potenciales, tales como code smells y security smells, antes de que el código entre en producción, lo cual puede ayudar a mejorar la calidad del software y asegurar la seguridad desde etapas tempranas del desarrollo (Walker y Cerny, 2020).

Una de las grandes ventajas del análisis estático es que puede integrarse fácilmente en el flujo de trabajo del desarrollo de software, utilizando herramientas que revisan automáticamente el código en busca de errores, malos olores de código y vulnerabilidades de seguridad. Herramientas como SonarQube, por ejemplo, son capaces de realizar revisiones automáticas y aplicar un conjunto de reglas predefinidas para detectar y reportar estos problemas, lo que facilita mucho el proceso de revisión de código y mejora continua (Bhalia y Thacker, 2020).

El análisis estático no solo se limita a la detección de problemas, sino que también incluye sugerencias para resolver estos problemas identificados. Durante un proceso de revisión de código, los revisores pueden sugerir cambios específicos para resolver los malos olores de código, y los desarrolladores pueden decidir implementar o ignorar estas sugerencias. Esto forma parte de un diálogo constructivo que busca mejorar la calidad del código y reducir la deuda técnica a largo plazo (Han et al., 2021).

1.1.3.1 Diferencias principales de análisis de código estático de IaC y el tradicional.

Contexto de infraestructura: Las herramientas para IaC están diseñadas específicamente para analizar configuraciones de infraestructura, no solo código de aplicación. Esto incluye la verificación de recursos como instancias de computación, redes, bases de datos, etc. (Walker y Cerny, 2020).

Lenguajes y formatos específicos: Las herramientas están adaptadas para trabajar con lenguajes y formatos de configuración como YAML, JSON, HCL, que son comunes en IaC, en lugar de lenguajes de programación tradicionales como Java, Python, o C++.

Enfoque en seguridad y cumplimiento: Hay un fuerte enfoque en la seguridad y el cumplimiento de normativas debido a la naturaleza crítica de las configuraciones de infraestructura.

1.1.4 Herramientas de análisis estático para scripts de IaC

Las herramientas de análisis estático en scripts de Infraestructura como Código examinan el código sin ejecutarlo para identificar errores, vulnerabilidades y malas prácticas. Ofrecen funcionalidades como detección de *code smells* y *security smells*, con informes detallados. Sin embargo, tienen limitaciones en la detección de ciertos problemas y contextos específicos. Es crucial seleccionar y configurar estas herramientas adecuadamente para garantizar la calidad y seguridad del código desplegado (Chiari et al., 2022).

ACID: Diseñada para evaluar la integridad, eficiencia y seguridad del código en entornos de Infraestructura como Código (IaC), ACID utiliza técnicas avanzadas para detectar vulnerabilidades y mejorar la calidad del código (Rahman et al., 2020).
BARREL: Esta herramienta facilita el análisis estático de scripts de IaC, proporcionando evaluaciones detalladas sobre la eficiencia y seguridad de las configuraciones de infraestructura (Brogi et al., 2015).
Checkov: Herramienta de código abierto que analiza configuraciones de IaC como Terraform, CloudFormation, Kubernetes y ARM para identificar errores, vulnerabilidades y malas prácticas, asegurando el cumplimiento de políticas de seguridad (BridgeCrew, 2024).
DeepIaC: Utiliza técnicas de aprendizaje profundo para evaluar scripts de IaC, mejorando la seguridad y la calidad del código mediante la detección automática de vulnerabilidades y errores de configuración (Borovits et al., 2020).
Foodcritic: Específica para scripts de Chef, Foodcritic analiza las recetas para detectar malas prácticas y errores, proporcionando recomendaciones para mejorar la calidad y seguridad del código (Schwarz et al., 2018).
GLITCH: Se enfoca en la identificación de vulnerabilidades y errores en scripts de IaC, ofreciendo análisis detallados y recomendaciones para mitigar riesgos y mejorar la seguridad de las configuraciones de infraestructura (Saavedra y Ferreira, 2022).
Hayha: Herramienta de análisis estático diseñada para identificar inconsistencias y posibles vulnerabilidades en configuraciones de IaC, ayudando a mantener una infraestructura segura y eficiente (Lepiller et al., 2021).

KICS: Analiza múltiples tipos de scripts de IaC, incluyendo Terraform, CloudFormation y Ansible, para encontrar vulnerabilidades y errores de configuración, mejorando la seguridad y eficiencia del código (KICS, 2024).

Puppeteer: Verifica la calidad y seguridad de scripts de configuración automatizados en entornos de IaC, asegurando que cumplan con las mejores prácticas y políticas de seguridad establecidas (Sharma et al., 2016).

RADON: Herramienta específica para Python, RADON proporciona métricas de calidad del código y detecta problemas de diseño, ayudando a los desarrolladores a mantener un código limpio y eficiente (Dalla Palma et al., 2022).

Rehearsal: Enfocada en la validación y prueba de scripts de IaC, Rehearsal asegura que las configuraciones sean correctas y seguras antes de su implementación en producción (Shambaugh et al., 2016).

SecureCode: Es una herramienta de análisis de seguridad que se integra en el ciclo de vida del desarrollo de software. Realiza análisis estático y dinámico del código fuente de manera continua, identificando y corrigiendo vulnerabilidades tempranamente. Esta integración permite a los desarrolladores mejorar la seguridad y calidad del software desde las primeras fases de desarrollo, reduciendo riesgos y fomentando prácticas de codificación segura. (Dai et al., 2020).

SLAC: Automatiza el análisis de seguridad y cumplimiento durante todo el ciclo de vida del desarrollo de software, asegurando que el código cumpla con las políticas y normas establecidas (Rahman et al., 2021).

SLIC: Evalúa la seguridad y eficiencia de los scripts de IaC, proporcionando informes detallados sobre posibles vulnerabilidades y recomendaciones de mejora (Rahman et al., 2019).

SNYK: Analiza el código fuente, dependencias y configuraciones para detectar vulnerabilidades, ofreciendo soluciones para mejorar la seguridad en aplicaciones y entornos de IaC (Snyk, 2024).

Sommelier: Herramienta de análisis estático que evalúa la calidad y seguridad de los scripts de IaC, identificando malas prácticas y errores que podrían comprometer la integridad de la infraestructura (Brogi et al., 2018).

SonarQube: Soporta múltiples lenguajes de programación y proporciona métricas de calidad del código, detectando bugs, vulnerabilidades y problemas de diseño para mantener un código limpio y seguro (SonarSource, 2024).

TAMA: Diseñada para entornos de IaC, TAMA se enfoca en la evaluación de la seguridad y eficiencia de las configuraciones, ayudando a los equipos de DevOps a mantener prácticas seguras y optimizadas (Hassan y Rahman, 2022).

Terrascan: Herramienta de código abierto para el análisis estático de scripts de Terraform, que escanea y detecta problemas de seguridad y cumplimiento, asegurando que las configuraciones sigan las mejores prácticas (Runterrascan.io, 2022).

Tfsec: Escanea archivos de configuración de Terraform para detectar vulnerabilidades y errores, proporcionando recomendaciones para mejorar la seguridad y conformidad de la infraestructura (Aquasecurity, 2024).

Trivy: Es una herramienta de análisis estático conocida principalmente como un escáner de vulnerabilidades de contenedores, pero también puede analizar configuraciones de IaC. Detecta problemas de seguridad y ofrece soluciones para mejorar la seguridad de las configuraciones (Aqua Security, 2024).

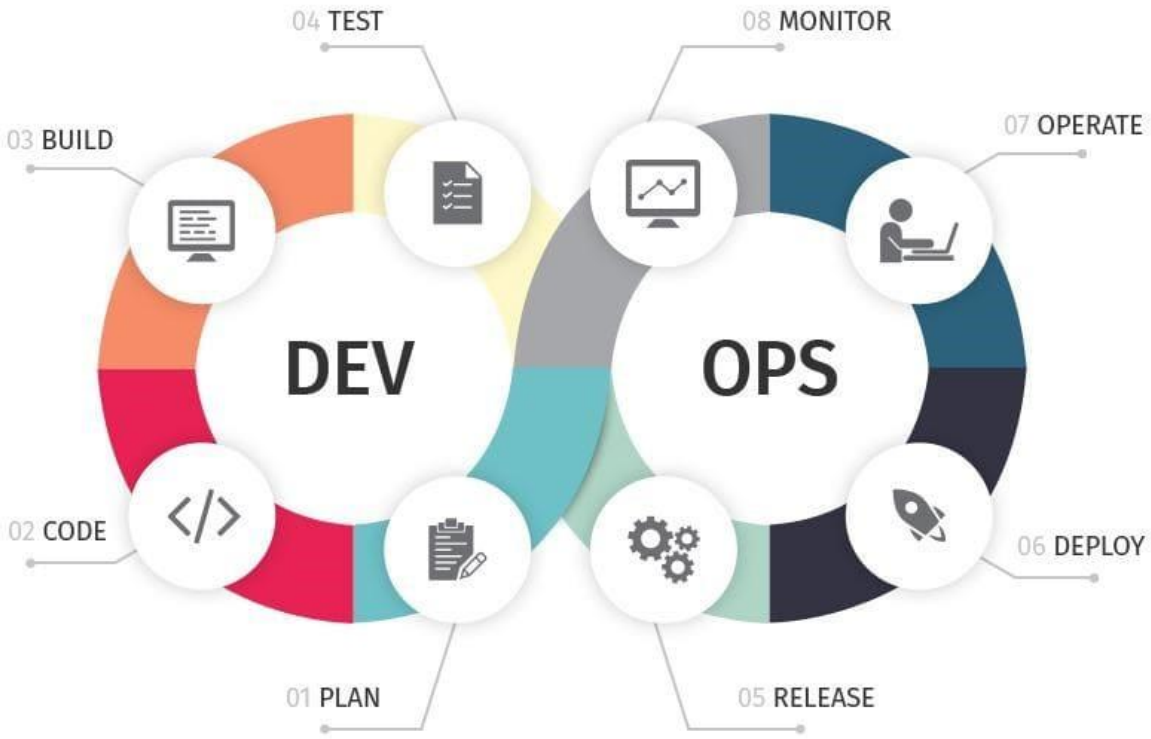
1.1.5 Objetivos en herramientas de análisis estático en scripts de IaC

- Detección de Errores de Configuración: Identificar configuraciones incorrectas en los scripts de IaC para evitar problemas durante la implementación.
- Seguridad y Conformidad: Detectar vulnerabilidades de seguridad y asegurar que los scripts cumplan con las políticas de conformidad y mejores prácticas de seguridad.
- Optimización de Recursos: Asegurar que los recursos de infraestructura se utilicen de manera eficiente y evitar desperdicios.
- Validación de Integridad y Consistencia: Verificar que los scripts mantengan la integridad y consistencia de la infraestructura provisionada.
- Mantenibilidad y Legibilidad del Código: Mejorar la calidad del código para facilitar su mantenimiento y asegurar que siga las mejores prácticas de codificación.
- Automatización e Integración Continua (CI/CD): Integrar el análisis estático en los pipelines de CI/CD para prevenir errores y vulnerabilidades antes de desplegar el código en producción.
- Detección de "Code Smells": Identificar patrones de código que pueden indicar problemas potenciales en el diseño o implementación de los scripts de IaC.
- Auditoría y Cumplimiento Normativo: Facilitar la auditoría de los scripts para asegurar el cumplimiento de normas y regulaciones aplicables.

1.1.6 DevOps

DevOps, una combinación de desarrollo (Dev) y operaciones (Ops), se centra en la colaboración entre equipos de desarrollo y operaciones para automatizar procesos, acelerar la entrega de software y mejorar la calidad del producto final (Guerriero et al., 2019). En el contexto de DevOps, la infraestructura como código (IaC) ha surgido como una práctica fundamental. IaC permite definir y gestionar la infraestructura de TI de forma programática, utilizando código en lugar de configuraciones manuales, lo que facilita la implementación, la escalabilidad y la reproducibilidad del entorno de infraestructura.

Figura *1*
Intersección DevOps

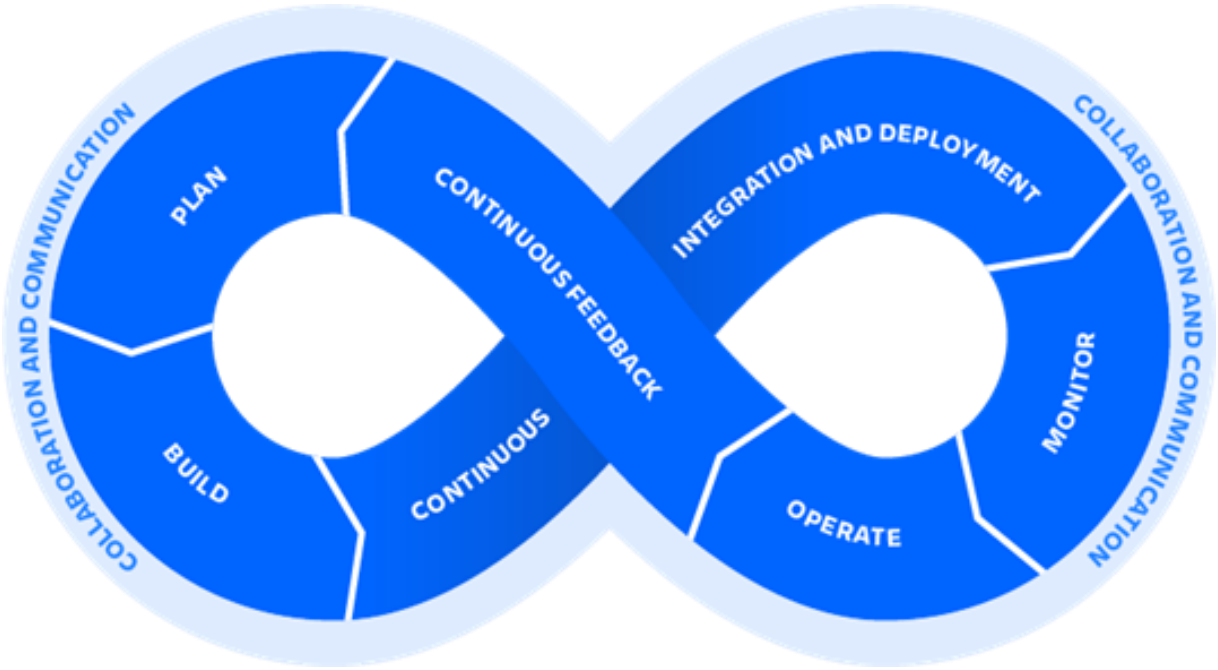


Fuente: Simform. (2023). DevOps lifecycle: 7 phases explained in detail. Retrieved from https://www.simform.com/devops-lifecycle-phases/

Una práctica de DevOps (Fig. 2) se define en forma de infinito debido a sus continuos pasos recurrentes en el proceso. (Atlassian, 2024) define los pasos principales del proceso de

DevOps como planificación, construcción e integración continua en el lado izquierdo seguidos del despliegue, operación y retroalimentación continua en el lado derecho. Esos pasos se siguen continuamente a lo largo de la práctica de DevOps y, tal como lo implica la forma infinita, no tienen un final estricto del proceso, lo que sí tienen es una retroalimentación continua y la colaboración del lado de operaciones con el lado del desarrollo y viceversa.

Figura 2
Ciclo de desarrollo en DevOps



Fuente*:* Adaptado de (Atlassian, 2024)

1.1.7 Code Smells

Los "code smells" son patrones de diseño o estructuras de código que pueden indicar posibles problemas en la calidad, mantenibilidad o seguridad del software. Estos indicadores son señales de alerta que sugieren la presencia de deficiencias en el diseño o la implementación del código, lo que puede conducir a errores, vulnerabilidades de seguridad o dificultades en la evolución y mantenimiento del software (Sharma et al., 2016).

Los code smells son identificados a través de la observación de ciertos atributos o características del código fuente que pueden indicar la necesidad de realizar mejoras o refactorizaciones. Algunos ejemplos comunes de code smells incluyen la duplicación de

código, métodos o clases excesivamente largos, acoplamiento excesivo entre componentes, entre otros (Rahman et al., 2021).

La detección y corrección temprana de *code smells* es fundamental para mejorar la calidad y la mantenibilidad del software, así como para prevenir posibles problemas en etapas posteriores del desarrollo.

1.1.8 Security Smells

Los *security smells* se definen como indicadores de posibles vulnerabilidades de seguridad en el código, que pueden manifestarse a través de malas prácticas de codificación, configuraciones inseguras o patrones de diseño propensos a ataques. Estos *smells* son señales de alerta que sugieren la presencia de debilidades en la seguridad del software, lo que podría exponerlo a riesgos de explotación por parte de actores malintencionados (Schwarz et al., 2018).

La identificación y corrección de *security smells* es crucial para fortalecer la seguridad de las aplicaciones y sistemas de software, ya que permite abordar posibles vulnerabilidades antes de que sean explotadas. Algunos ejemplos de security smells incluyen el uso de contraseñas codificadas de forma insegura, la falta de validación de entradas de usuario o la exposición de información sensible (Sharma et al., 2016).

Es fundamental realizar análisis de seguridad exhaustivos y utilizar herramientas especializadas para detectar y mitigar los *security smells* en el código, con el fin de proteger la integridad y confidencialidad de los sistemas de software (Rahman et al., The seven sins: Security smells in infrastructure as code scripts, 2019).

1.2 Marco Contextual

La Infraestructura como Código (IaC) se ha convertido en una práctica esencial en la gestión moderna de infraestructuras de TI, permitiendo la automatización y estandarización de entornos a través de scripts. En el contexto de DevOps, IaC facilita la integración y entrega continua (CI/CD), mejorando la eficiencia y reduciendo errores humanos en la configuración de infraestructuras. Sin embargo, la creciente complejidad de los entornos de TI y la diversidad de tecnologías presentan desafíos significativos en la detección y corrección de vulnerabilidades, errores de configuración y "code smells" en los scripts de IaC.

La necesidad de herramientas de análisis estático para scripts de IaC surge en respuesta a estos desafíos. Estas herramientas permiten inspeccionar el código sin necesidad de ejecutarlo, identificando posibles errores y vulnerabilidades de manera temprana en el ciclo de desarrollo. En entornos DevOps, donde la rapidez y precisión son cruciales, la integración de herramientas de análisis estático ayuda a asegurar la calidad y seguridad del código antes de su implementación en producción. Sin embargo, la variedad de herramientas disponibles y sus diferentes enfoques y capacidades requieren una evaluación sistemática para determinar cuáles son las más efectivas y adecuadas para distintas necesidades.

La presente investigación, de tipo teórica-descriptiva y analítica-descriptiva, utiliza métodos de investigación documental y bibliográfica para evaluar el estado actual de las herramientas de análisis estático para scripts de IaC. El objetivo es proporcionar una guía basada en evidencia que facilite la selección informada de estas herramientas, mejorando así la calidad y seguridad de las infraestructuras gestionadas mediante IaC. Al identificar las técnicas y metodologías más efectivas, esta monografía busca contribuir a la optimización de las prácticas de DevOps y a la protección de las infraestructuras de TI contra posibles vulnerabilidades y errores.

CAPITULO II

2 DIAGNOSTICO

2.1 Introducción

En el contexto actual de la infraestructura como código (IaC), la calidad de los scripts de configuración desempeña un papel fundamental en la seguridad, eficiencia y confiabilidad de los sistemas de software. La detección temprana de posibles vulnerabilidades, errores de configuración y *code smells* en los *scripts* de IaC es esencial para prevenir fallos en la infraestructura y posibles brechas de seguridad.

El uso de herramientas de análisis estático para scripts de IaC permite a los equipos de DevOps automatizar la revisión de código, identificando problemas potenciales antes de que estos sean desplegados en producción. Estas herramientas no solo ayudan a mantener altos estándares de calidad del código, sino que también mejoran la productividad al reducir el tiempo necesario para las revisiones manuales. Dado el creciente número de herramientas disponibles en el mercado, es fundamental contar con criterios claros y preguntas guía que faciliten su selección y uso efectivo en diferentes contextos organizacionales.

En este contexto, el presente diagnóstico se enfoca en investigar las herramientas de análisis estático para scripts de Infraestructura como Código (IaC) y plantear preguntas guía para una correcta elección. A través de una revisión exhaustiva de la literatura existente y un análisis comparativo de las herramientas y técnicas disponibles, este diagnóstico busca identificar patrones, tendencias y brechas en el campo del análisis estático en scripts de IaC.

2.1.1 Procesamiento y Análisis de Datos

2.1.1.1 Herramientas de análisis estático para scripts de IaC

Se realizó una exhaustiva revisión de las herramientas de análisis estático para scripts de Infraestructura como Código (IaC), abarcando tanto las respaldadas por investigaciones académicas como las adoptadas en la industria tecnológica. Estas herramientas se organizaron en dos grupos: aquellas con respaldo académico y las populares en la industria, ofreciendo una visión completa de las opciones disponibles.

La tabla 1 presenta herramientas respaldadas por publicaciones académicas, indicando el estudio que las respalda y el nombre de la herramienta.

Tabla 1 Herramientas respaldadas por publicaciones académicas

Estudio que lo respalda	Nombre de la herramienta
(Rahman et al., 2020)	ACID
(Brogi et al., 2015)	BARREL
(Borovits et al., 2020)	DeepIaC
(Schwarz et al., 2018)	Foodcritic
(Saavedra y Ferreira, 2022)	GLITCH
(Lepiller et al., 2021)	Hayha
(Sharma et al., 2016)	Puppeteer
(Dalla Palma et al., 2022)	RADON
(Shambaugh et al., 2016)	Rehearsal
(Dai et al., 2020)	SecureCode
(Rahman et al., 2021)	SLAC
(Rahman et al., 2019)	SLIC
(Brogi et al., 2018)	Sommelier
(Hassan y Rahman, 2022)	TAMA

Fuente: Elaboración propia

La tabla 2 presenta herramientas populares en la industria tecnológica, señalando la fuente de información y el nombre de la herramienta.

Tabla 2 Herramientas populares en la industria de la tecnología

Fuente de Información	Nombre de la herramienta
(BridgeCrew, 2024)	Checkov
(KICS, 2024)	KICS
(Snyk, 2024)	Snyk laC
(SonarSource, 2024)	SonarQube
(Runterrascan.io, 2022)	Terrascan
(Aquasecurity, 2024)	Tfsec
(Aqua Security, 2024)	Trivy

Fuente: Elaboración propia

2.1.1.2 Características técnicas de herramientas de análisis estático para scripts de IaC

Las características técnicas se refieren a las capacidades y métodos de las herramientas para realizar el análisis de código estático. Estas incluyen:

- Lenguaje soportado

- Técnicas de análisis
- Objetivo de análisis
- Categorización en el contexto de IaC

Las características técnicas de cada herramienta de análisis estático para scripts de IaC evaluadas en esta investigación se describen en el **ANEXO A, B y C**

2.1.1.3 Características operacionales de herramientas de análisis estático para scripts en IaC.

Las características operativas afectan cómo se implementa y se utiliza la herramienta de análisis estático. Estos aspectos incluyen:

- Integración con IDEs, sistema de control y Pipelines de CI/CD
- Licencia o costo
- Documentación y soporte

Las características de uso de cada herramienta de análisis estático para scripts de IaC evaluadas en esta investigación se describen en el **ANEXO D, E** y **F.**

2.1.1.4 Criterio de evaluación para plantear preguntas para la selección de herramientas de análisis estático en scripts de IaC.

El criterio de evaluación implica una revisión minuciosa de diversas características y funcionalidades para garantizar la eficacia y adecuación de las herramientas al entorno de trabajo. Las preguntas abordan criterios como facilidad de uso, capacidad de detección y reporte de errores, compatibilidad con diferentes lenguajes de IaC e integración con otros componentes del entorno de desarrollo y operaciones.

Se identificaron ocho criterios de evaluación para establecer la guía de preguntas:

- Compatibilidad con Lenguajes y Plataformas de IaC
- Modelo de Licencia y Costos
- Integración con Entornos de Desarrollo
- Facilidad de Uso y Documentación
- Capacidades de Análisis y Reportes
- Soporte y Actualizaciones
- Adaptabilidad y Personalización

Estos criterios proporcionan un marco sólido para estructurar la guía de preguntas, asegurando que se aborden todos los aspectos relevantes para la elección adecuada de herramientas de análisis estático en el contexto de Infraestructura como Código.

2.1.2 Tabulación y Codificación de Datos

2.1.2.1 Características técnicas de herramientas de análisis estático para scripts de IaC

- Lenguaje soportado

Tabla 3 Lenguajes soportado por herramientas de análisis estático en scripts de IaC

Herramienta	Lenguaje Soportado
ACID	Puppet
BARREL	TOSCA
Checkov	Terraform, CloudFormation, Kubernetes, Docker,
DeepIaC	Ansible
Foodcritic	Chef
GLITCH	Ansible, Chef y Puppet
Hayha	CloudFormation
KICS	Terraform, Kubernetes, Docker, CloudFormation, Ansible
Puppeteer	Puppet
RADON	Ansible
Rehearsal	Puppet
SecureCode	Ansible
SLAC	Ansible y Chef
SLIC	Puppet.
SNYK IaC	Terraform, CloudFormation, Kubernetes, ARM, Docker
Sommelier	TOSCA
SonarQube	Terraform, CloudFormation, Kubernetes, Docker
TAMA	Ansible
Terrascan	Terraform, CloudFormation, Kubernetes, Docker
Tfsec	Terraform, CloudFormation, Kubernetes, ARM
Trivy	Kubernetes, Docker, Terraform

Fuente: *Elaboración propia*

Interpretación: La interpretación se deriva de la tabla 3 y del Anexo A.

- Las herramientas Checkov, KICS, Snyk IaC, Terrascan y SonarQube soportan múltiples lenguajes y plataformas, lo que facilita su integración en entornos de DevOps. Esta versatilidad permite detectar configuraciones inseguras y vulnerabilidades en diversas infraestructuras, mejorando la seguridad en todo el ciclo de vida del desarrollo y despliegue.

- Por otro lado, herramientas académicas como ACID, BARREL, DeepIaC, Foodcritic, GLITCH, Hayha, Puppeteer, RADON, Rehearsal, SecureCode, SLAC, SLIC, Sommelier y TAMA se especializan en lenguajes y plataformas específicas. Estas herramientas proporcionan análisis detallados y metodologías innovadoras, cruciales para entornos específicos y proyectos de investigación.

➢ Técnica de análisis

` Las técnicas utilizadas por las herramientas de análisis estático para scripts de Infraestructura como Código (IaC) fueron identificadas durante la revisión bibliográfica realizada en la investigación. Se destacaron las siguientes siete técnicas:

- Algoritmos Ad hoc
- Algoritmos de Patrones
- Análisis de Gráficos
- Aprendizaje Profundo y Motor de IA
- Expresiones Regulares
- Minería de Datos
- Solver SMT

En la tabla 4 se presentan las técnicas de análisis que utiliza cada herramienta en base a la información obtenida del Anexo B.

Tabla 4 Técnicas de herramientas de análisis estático en scripts de IaC

Herramienta	Técnicas de análisis
ACID	Expresiones regulares
BARREL	Análisis de gráficos
Checkov	Expresiones regulares, Algoritmos de patrones
DeepIaC	Aprendizaje profundo y Motor de IA
Foodcritic	Expresiones regulares, Algoritmos de patrones
GLITCH	Solver SMT, Expresiones regulares
Hayha	Análisis de Gráficos
KICS	Expresiones regulares, Algoritmos de patrones
Puppeteer	Expresiones regulares, Algoritmos de patrones
RADON	Expresiones Regulares, Algoritmos de Patrones
Rehearsal	Solver SMT
SecureCode	Expresiones Regulares, Algoritmos de Patrones
SLAC	Expresiones Regulares, Algoritmos de Patrones
SLIC	Expresiones regulares

Snyk IaC	Expresiones regulares, Aprendizaje profundo
Sommelier	Expresiones regulares, Algoritmos de patrones
SonarQube	Expresiones Regulares, Algoritmos de Patrones
TAMA	Expresiones regulares, Algoritmos de patrones
Terrascan	Expresiones Regulares, Algoritmos de Patrones
Tfsec	Expresiones regulares, Algoritmos de patrones
Trivy	Expresiones regulares, Algoritmos de patrones

Fuente: *Elaboración propia*

Interpretación: La interpretación se deriva de la tabla 4 y del Anexo B.

- Las herramientas populares como Checkov, KICS, Snyk IaC, SonarQube, Terrascan, Tfsec y Trivy utilizan expresiones regulares y algoritmos de patrones para detectar configuraciones inseguras y vulnerabilidades en scripts de IaC. Las herramientas académicas como ACID, BARREL, DeepIaC, Foodcritic, GLITCH, Hayha, Puppeteer, RADON, Rehearsal, SecureCode, SLAC, SLIC, Sommelier y TAMA emplean técnicas variadas, incluyendo aprendizaje profundo y análisis de gráficos, mejorando la seguridad y calidad de los scripts de IaC.
- Herramientas como Puppeteer, SLAC y Sommelier combinan expresiones regulares y algoritmos de patrones, permitiendo detectar una amplia gama de problemas, desde errores de sintaxis hasta configuraciones inseguras complejas.
- Herramientas como GLITCH, Rehearsal y DeepIaC utilizan técnicas específicas como Solver SMT y aprendizaje profundo para realizar análisis precisos y automatizados de scripts de IaC, garantizando que los scripts sean consistentes y libres de errores, mejorando la calidad y seguridad del código.

➢ Objetivos de análisis

Tabla 5 Objetivo de análisis de las herramientas de análisis estático en IaC

Herramienta	Objetivos de análisis
ACID	Detección de Errores de Configuración, Seguridad y Conformidad, Mantenibilidad y Legibilidad del Código
BARREL	Validación de Integridad y Consistencia, Mantenibilidad y Legibilidad del Código, Automatización e Integración Continua (CI/CD)
Checkov	Detección de Errores de Configuración, Seguridad y Conformidad, Automatización e Integración Continua (CI/CD), Auditoría y Cumplimiento Normativo
DeepIaC	Mantenibilidad y Legibilidad del Código, Detección de "Code Smells", Validación de Integridad y Consistencia
Foodcritic	Detección de Errores de Configuración, Mantenibilidad y Legibilidad del Código, Detección de "Code Smells"

GLITCH	Seguridad y Conformidad, Detección de "Code Smells", Validación de Integridad y Consistencia, Auditoría y Cumplimiento Normativo
Hayha	Detección de Errores de Configuración, Seguridad y Conformidad, Validación de Integridad y Consistencia, Automatización e Integración Continua (CI/CD)
KICS	Detección de Errores de Configuración, Seguridad y Conformidad, Optimización de Recursos, Automatización e Integración Continua (CI/CD), Auditoría y Cumplimiento Normativo
Puppeteer	Detección de Errores de Configuración, Seguridad y Conformidad, Mantenibilidad y Legibilidad del Código, Detección de "Code Smells", Automatización e Integración Continua (CI/CD)
RADON	Detección de Errores de Configuración, Seguridad y Conformidad, Mantenibilidad y Legibilidad del Código, Automatización e Integración Continua (CI/CD), Detección de "Code Smells"
Rehearsal	Detección de Errores de Configuración, Validación de Integridad y Consistencia
SecureCode	Detección de Errores de Configuración, Seguridad y Conformidad, Automatización e Integración Continua (CI/CD), Detección de "Code Smells"
SLAC	Detección de Errores de Configuración, Seguridad y Conformidad, Mantenibilidad y Legibilidad del Código, Detección de "Code Smells"
SLIC	Seguridad y Conformidad, Detección de "Code Smells", Auditoría y Cumplimiento Normativo
Snyk IaC	Seguridad y Conformidad, Automatización e Integración Continua (CI/CD), Detección de "Code Smells"
Sommelier	Detección de Errores de Configuración, Validación de Integridad y Consistencia, Auditoría y Cumplimiento Normativo
SonarQube	Detección de Errores de Configuración, Seguridad y Conformidad, Mantenibilidad y Legibilidad del Código, Automatización e Integración Continua (CI/CD), Detección de "Code Smells"
TAMA	Detección de Errores de Configuración, Mantenibilidad y Legibilidad del Código, Detección de "Code Smells"
Terrascan	Seguridad y Conformidad, Automatización e Integración Continua (CI/CD), Auditoría y Cumplimiento Normativo
Tfsec	Detección de Errores de Configuración, Seguridad y Conformidad, Automatización e Integración Continua (CI/CD)
Trivy	Detección de Errores de Configuración, Seguridad y Conformidad, Automatización e Integración Continua (CI/CD)

Fuente: *Elaboración propia*

Interpretación: La interpretación se deriva de la tabla 5 y el apartado 1.1.5

- Las herramientas de análisis estático de IaC, tanto académicas como populares, comparten objetivos clave: detección de errores de configuración, seguridad y conformidad, mejora de la mantenibilidad y legibilidad del código, y validación de la integridad y consistencia de los scripts. Herramientas populares como Checkov,

KICS y SonarQube también destacan por su automatización e integración continua (CI/CD) y auditoría y cumplimiento normativo.

- Herramientas como ACID y KICS identifican configuraciones incorrectas y vulnerabilidades de seguridad, asegurando el cumplimiento de políticas de seguridad y mejores prácticas. DeepIaC y SonarQube detectan "code smells" y malas prácticas de codificación, facilitando la corrección de problemas y mejorando la calidad del código.

➢ Categoría de IaC

Tabla 6 Categorización de herramientas de análisis estático en scripts de IaC

Herramienta	Categoría en IaC
ACID	Gestión de configuración
BARREL	Aprovisionamiento de infraestructura
Checkov	Aprovisionamiento de infraestructura, gestión de configuración
DeepIaC	Gestión de configuración
Foodcritic	Gestión de configuración
GLITCH	Gestión de configuración
Hayha	Gestión de configuración
KICS	Aprovisionamiento de infraestructura, gestión de configuración
Puppeteer	Gestión de configuración
RADON	Gestión de configuración
Rehearsal	Gestión de configuración
SecureCode	Gestión de configuración
SLAC	Gestión de configuración
SLIC	Gestión de configuración
SNYK IaC	Gestión de configuración
Sommelier	Aprovisionamiento de infraestructura
SonarQube	Gestión de configuración
TAMA	Gestión de configuración
Terrascan	Aprovisionamiento de infraestructura
Tfsec	Aprovisionamiento de infraestructura
Trivy	Gestión de configuración

Fuente: *Elaboración propia*

Interpretación: La interpretación se basa en la tabla 6 y la información del Anexo C.

- Las herramientas académicas como ACID, DeepIaC y RADON se centran en la gestión de configuración, identificando defectos y vulnerabilidades en scripts específicos como Puppet y Ansible.

- Por otro lado, las herramientas populares en la industria, como Checkov, KICS y Terrascan, sobresalen en el aprovisionamiento de infraestructura. Estas herramientas soportan múltiples plataformas y lenguajes, permitiendo verificar configuraciones y detectar vulnerabilidades de manera integral.
- Herramientas híbridas, como KICS y SonarQube, abarcan tanto la gestión de configuración como el aprovisionamiento de infraestructura. KICS, por ejemplo, analiza configuraciones de Terraform y CloudFormation, además de escanear scripts de Ansible, Chef y Puppet. Esta versatilidad permite a los usuarios abordar múltiples aspectos de IaC con una sola herramienta, optimizando los procesos de desarrollo y operaciones.

2.1.2.2 Características operacionales

- Tipo de Integración

Tabla 7 Integración de herramientas de análisis estático en IaC con IDEs, sistemas de sontrol de versiones y Pipelines de CI/CD

Herramienta	Integración con IDEs, Sistema de control y Pipelines de CI/CD
ACID	No se menciona
BARREL	No se menciona
Checkov	Se integra con IDEs, sistemas de control de versiones y pipelines de CI/CD
DeepIaC	No se menciona
Foodcritic	No se menciona
GLITCH	Se puede integrar
Hayha	No menciona
KICS	Se integra con IDEs, sistemas de control de versiones y pipelines de CI/CD
Puppeteer	GitHub y herramientas de CI/CD.
RADON	GitHub, pipelines de CI/CD.
Rehearsal	No menciona
SecureCode	CI/CD como Travis CI
SLAC	No menciona
SLIC	No menciona
SNYK IaC	Se integra con IDEs, sistemas de control de versiones y pipelines de CI/CD
Sommelier	No menciona
SonarQube	Se integra con IDEs, sistemas de control de versiones y plataformas CI/CD
TAMA	No menciona
Terrascan	Se integra con IDEs, sistemas de control de versiones y pipelines de CI/CD
Tfsec	Se integra con IDEs, sistemas de control de versiones y pipelines de CI/CD
Trivy	Se integra con IDEs, sistemas de control de versiones y pipelines de CI/CD

Fuente: *Elaboración propia*

Interpretación: La interpretación se deriva de la tabla 7 y del Anexo D.

- Checkov, KICS, Snyk IaC, SonarQube, Terrascan, Tfsec y Trivy se integran completamente con IDEs, sistemas de control de versiones y pipelines de CI/CD. Permiten un análisis continuo y en tiempo real, proporcionando retroalimentación inmediata sobre la calidad y seguridad del código durante el desarrollo y despliegue. La integración con plataformas como GitHub, GitLab, Jenkins y Visual Studio Code asegura configuraciones y aprovisionamiento de infraestructura seguros y conformes a las normativas.
- GLITCH y Puppeteer presentan una integración parcial con sistemas de control de versiones y CI/CD. GLITCH se adapta a diversos entornos de desarrollo mediante scripts personalizados. Puppeteer analiza repositorios en GitHub y se ejecuta en pipelines de CI/CD para verificar la calidad del código de configuración automáticamente. Aunque no se mencionan integraciones directas con IDEs, estas herramientas mejoran la consistencia y predictibilidad de las configuraciones de infraestructura.
- Herramientas como ACID, BARREL, DeepIaC, Foodcritic, Hayha, RADON, Rehearsal, SLAC, SLIC, Sommelier y TAMA no presentan integraciones específicas con IDEs, sistemas de control de versiones o pipelines de CI/CD. Estas herramientas se centran en la detección de errores, análisis de configuraciones y aseguramiento de la calidad del código. Su diseño modular sugiere que podrían adaptarse mediante scripts personalizados o herramientas adicionales para mejorar la consistencia y seguridad en la gestión de IaC.

➢ Características de licencia o costo

Tabla 8 Licencia o costo de herramientas de análisis estático en scripts de IaC

Herramienta	Tipo de licencia
ACID	Código abierto
BARREL	Código abierto
Checkov	Código abierto y gratuita
DeepIaC	Código abierto
Foodcritic	Código abierto
GLITCH	Código abierto
Hayha	Código abierto
KICS	Código abierto y gratuita
Puppeteer	Código abierto

RADON	Código abierto
Rehearsal	Código abierto
SecureCode	No se menciona
SLAC	Código abierto
SLIC	Código abierto
SNYK IaC	Freemium
Sommelier	Código abierto y gratuita
SonarQube	Freemium
TAMA	Código abierto
Terrascan	Código abierto y gratuita
Tfsec	Código abierto y gratuita
Trivy	Código abierto y gratuita

Fuente: *Elaboración propia*

Interpretación: La interpretación se deriva de la tabla 9 y del Anexo E.

- La mayoría de las herramientas de análisis estático de scripts de IaC, como Checkov, KICS, Puppeteer, RADON, Sommelier, Terrascan, Tfsec y Trivy, son de código abierto y gratuitas. Estas herramientas se distribuyen bajo licencias como Apache 2.0 o Creative Commons, lo que permite su uso, modificación y distribución sin costo. Este enfoque fomenta la colaboración y contribución de la comunidad de desarrolladores, asegurando la mejora continua de las herramientas.
- Algunas herramientas, como Snyk IaC y SonarQube, operan bajo un modelo de licencia Freemium. Estas ofrecen una versión gratuita con funcionalidades básicas y versiones pagadas que proporcionan características avanzadas y soporte adicional. Este modelo permite a los usuarios empezar a utilizar la herramienta sin costo, con la opción de pagar por funcionalidades adicionales según las necesidades de sus proyectos.
- Varias herramientas, como ACID, BARREL, DeepIaC, Foodcritic, GLITCH, Hayha, Rehearsal, SLAC y SLIC, fueron desarrolladas en un contexto académico y están disponibles gratuitamente para uso educativo y de investigación. No se especifican detalles sobre integraciones comerciales o modelos de licencia específicos, aunque generalmente se distribuyen sin costo para fomentar la investigación y el uso educativo. SecureCode, desarrollada por IBM, probablemente se utiliza internamente y no se menciona si tiene un modelo Freemium o algún costo asociado.

- Documentación y soporte

Tabla 9 Documentación y soporte de herramientas de análisis estático en scripts de IaC

Herramienta	Soporte	Enlace de la documentación/repositorio
ACID	No se menciona	
Barrel	Soporte y documentación	*URL: https://github.com/di-unipi-socc/barrel*
Checkov	Soporte y documentación	*URL: https://github.com/bridgecrewo/checkov*
DeepIaC	Soporte y documentación	*URL: https://github.com/SODALITE-EU/defect-prediction/tree/master/ansible*
Foodcritic	Soporte y documentación	*URL: https://github.com/Foodcritic/foodcritic*
GLITCH	Soporte y documentación	*URL: https://github.com/sr-lab/GLITCH*
Häyhä	Soporte y documentación	*URL: https://gitlab.com/rose-yale/hayha*
KICS	Soporte y documentación	*URL: https://checkmarx.com/product/opensource/kics-open-source-infrastructure-as-code-project/*
Puppeteer	Soporte y documentación	*URL: https://pptr.dev/*
RADON	Soporte y documentación	*URL: https://pypi.org/project/repositories-collector/*
Rehearsal	Soporte y documentación	*URL: https://github.com/plasma-umass/Rehearsal*
SecureCode	No se menciona	
SLAC	No se menciona	
SLIC	Soporte y documentación	*URL: https://github.com/rayhanur-rahman/SLIC-Ansible*
Snyk IaC	Soporte y documentación	*URL: https://docs.snyk.io/*
Sommelier	Soporte y documentación	*URL: https://github.com/di-unipi-socc/Sommelier*
SonarQube	Soporte y documentación	*URL: https://github.com/SonarSource/sonar-iac*
TAMA	Soporte y documentación	*URL: https://github.com/akondrahman/IaCTesting*
Terrascan	Soporte y documentación	*URL: https://www.tenable.com/terrascan*
Tfsec	Soporte y documentación	*URL: https://aquasecurity.github.io/tfsec/latest/*
Trivy	Soporte y documentación	*URL: https://aquasecurity.github.io/trivy/v0.51/*

Fuente: *Elaboración propia*

Interpretación: La interpretación se basa en la tabla 10 y la información del Anexo F.

- Las herramientas de análisis estático para scripts de IaC, como Checkov, KICS, SonarQube, Trivy y Tfsec, se destacan por ofrecer documentación y soporte robusto. Estas herramientas proporcionan guías de instalación, ejemplos prácticos y documentación técnica detallada en sus repositorios de GitHub y páginas oficiales. Además, cuentan con una comunidad activa que contribuye con soluciones y actualizaciones, facilitando su uso e implementación, así como la resolución de problemas, lo cual es crucial al seleccionar herramientas para proyectos que requieren soporte continuo y eficiente.

- Por otro lado, herramientas como SLAC y SecureCode no especifican la disponibilidad de documentación formal o soporte extensivo. SLAC, desarrollada en un contexto académico, se distribuye con el código fuente en línea, lo que puede requerir una mayor adaptación por parte del usuario. SecureCode, desarrollada por IBM Research, parece ser una herramienta interna sin recursos de apoyo público, limitando su accesibilidad y utilidad para usuarios externos. La falta de soporte formal en estas herramientas puede ser una desventaja significativa para los usuarios que buscan una integración y soporte más accesibles.
- Al elegir una herramienta de análisis estático de scripts de IaC, es esencial considerar la disponibilidad de documentación detallada y soporte comunitario. Herramientas con documentación extensa y una comunidad activa, como Checkov y KICS, son más accesibles y fáciles de implementar, lo que asegura una integración eficiente y un soporte continuo, mejorando su efectividad en proyectos de IaC.

2.1.3 Análisis y Discusión de Resultados

2.1.3.1 Herramientas de análisis estático más utilizadas en scripts de Infraestructura como Código (IaC).

Tabla 10 Herramientas de análisis estático para scripts de IaC

Origen	Nombre de la herramienta	Cantidad
Herramientas respaldadas por publicaciones académicas	ACID, BARREL, DeepIaC, Foodcritic, GLITCH, Hayha, Puppeteer, RADON, Rehearsal, SecureCode, SLAC, SLIC, Sommelier TAMA	14
Herramientas populares en la industria de la tecnología	Checkov, KICS, SNYK IaC, SonarQube, Terrascan, Tfsec, Trivy	7

Fuente: *Elaboración propia*

Herramientas respaldadas por publicaciones académicas: Incluyen 14 herramientas como ACID, DeepIaC, y SecureCode, destacadas por su enfoque en investigación y desarrollo.

Herramientas populares en la industria de la tecnología: Incluyen 7 herramientas como Checkov, KICS, y SonarQube, valoradas por su integración en pipelines de CI/CD y facilidad de uso.

Este análisis proporciona una base sólida para elegir herramientas de análisis estático según necesidades específicas en entornos académicos o industriales.

2.1.3.2 Clasificación de herramientas de análisis estático para scripts de Infraestructura como Código (IaC) según sus características técnicas

Se presenta un resumen de los resultados de la clasificación de herramientas de análisis estático para scripts de IaC según sus características técnicas. Estas características incluyen el lenguaje soportado, las técnicas de análisis empleadas, el objetivo del análisis y su categorización en el contexto de IaC. Esta clasificación ayuda a mostrar los resultados de manera clara y estructurada, facilitando la elección adecuada según las necesidades específicas de los usuarios.

Tabla 11 Características técnicas de herramientas de análisis estático en scripts de IaC

Herramienta	Lenguaje Soportado	Técnicas de análisis	Objetivo de análisis	Categorización en el contexto de IaC
ACID	Puppet	Expresiones regulares	Detección de Errores de Configuración	Gestión de configuración
BARREL	TOSCA	Análisis de gráficos	Seguridad y Conformidad	Aprovisionamiento de infraestructura
Checkov	Terraform, CloudFormation, Kubernetes, Docker, etc.	Expresiones regulares, Algoritmos de patrones	Mantenibilidad y Legibilidad del Código	Aprovisionamiento de infraestructura, gestión de configuración
DeepIaC	Ansible	Aprendizaje profundo y Motor de IA	Validación de Integridad y Consistencia.	Gestión de configuración
Foodcritic	Chef	Expresiones regulares, Algoritmos de patrones	Mantenibilidad y Legibilidad del Código.	Gestión de configuración
GLITCH	Ansible, Chef y Puppet	Solver SMT, Expresiones regulares	Automatización e Integración Continua (CI/CD).	Gestión de configuración
Hayha	AWS CloudFormation	Análisis de Gráficos	Detección de Errores de Configuración	Gestión de configuración
KICS	Terraform, Kubernetes, Docker, CloudFormation, Ansible,	Expresiones regulares, Algoritmos de patrones	Seguridad y Conformidad	Aprovisionamiento de infraestructura, gestión de configuración
Puppeteer	Puppet	Expresiones regulares, Algoritmos de patrones	Automatización e Integración Continua (CI/CD)	Gestión de configuración
RADON	Ansible (YAML)	Expresiones regulares, Algoritmos de Patrones	Auditoría y Cumplimiento Normativo	Gestión de configuración
Rehearsal	Puppet	Solver SMT	Mantenibilidad y Legibilidad del Código	Gestión de configuración

SecureCode	Ansible	Expresiones regulares, Algoritmos de Patrones	Detección de "Code Smells"	Gestión de configuración
SLAC	Ansible y Chef	Expresiones Regulares, Algoritmos de Patrones	Validación de Integridad y Consistencia	Gestión de configuración
SLIC	Puppet.	Expresiones regulares	Detección de Errores de Configuración	Gestión de configuración
SNYK IaC	Terraform, CloudFormation, Kubernetes, ARM, Docker	Expresiones regulares, Aprendizaje profundo	Mantenibilidad y Legibilidad del Código	Gestión de configuración
Sommelier	TOSCA	Expresiones regulares, Algoritmos de patrones	Detección de "Code Smells"	Aprovisionamiento de infraestructura
SonarQube	Terraform, CloudFormation, Kubernetes, Docker	Expresiones Regulares, Algoritmos de Patrones	Seguridad y Conformidad	Gestión de configuración
TAMA	Ansible	Expresiones regulares, Algoritmos de patrones	Detección de "Code Smells"	Gestión de configuración
Terrascan	Terraform, CloudFormation, Kubernetes, Docker	Expresiones Regulares, Algoritmos de Patrones	Validación de Integridad y Consistencia	Aprovisionamiento de infraestructura
Tfsec	Terraform, CloudFormation, Kubernetes, ARM	Expresiones regulares, Algoritmos de patrones	Auditoría y Cumplimiento Normativo	Aprovisionamiento de infraestructura
Trivy	Kubernetes, Docker, Terraform	Expresiones regulares, Algoritmos de patrones	Detección de Errores de Configuración	Gestión de configuración

Fuente: Elaboración propia

- **Proceso de Análisis**

  Se evaluó cada herramienta en función de las categorías mencionadas: lenguajes soportados, técnicas de análisis, objetivos de análisis y su categorización en el contexto de IaC. Se recopiló información detallada y se organizó en tablas comparativas para facilitar la comprensión de las capacidades y limitaciones de cada herramienta.

- **Interpretación Crítica**
  - **Versatilidad y Popularidad:** Herramientas como Checkov, KICS y SonarQube son altamente versátiles y populares debido a su capacidad para soportar múltiples lenguajes y plataformas. La habilidad para detectar configuraciones inseguras y vulnerabilidades las hacen esenciales para los equipos de DevOps.

- **Especialización académica:** Herramientas como ACID, DeepIaC y RADON están especializadas en lenguajes y plataformas específicas, ofreciendo análisis detallados y metodologías innovadoras que son cruciales para entornos específicos y proyectos de investigación. Utilizan técnicas avanzadas como el aprendizaje profundo para mejorar la calidad y mantenibilidad de las configuraciones de IaC.
- **Detección de "Code Smells" y Mejora de la Calidad del Código:** Herramientas como DeepIaC, Foodcritic y SonarQube se destacan en la detección de "code smells" y malas prácticas de codificación. Estas herramientas facilitan la corrección de problemas que afectan la calidad del código, asegurando que los scripts sean fáciles de mantener y actualizar, reduciendo el riesgo de errores futuros.
- **Enfoque integral en seguridad:** La seguridad y conformidad son objetivos primordiales para herramientas como Checkov, KICS y SonarQube, que ofrecen soluciones robustas para detectar vulnerabilidades y asegurar que los scripts cumplan con las normativas de seguridad. Esto es esencial para prevenir problemas durante la implementación y operación de la infraestructura.

2.1.3.3 Clasificación de herramientas de análisis estático para scripts de Infraestructura como Código según sus características operacionales.

Se presentan los resultados de la clasificación de herramientas de análisis estático para scripts de IaC según su integración con IDEs, sistemas de control de versiones y pipelines de CI/CD, su licencia o costo, y la disponibilidad de documentación y soporte. Esta clasificación facilita la elección adecuada según las necesidades específicas de los usuarios.

Tabla 12 Características operacionales de herramientas de análisis estático en scripts de IaC

Herramienta	Integración con IDEs, Sistema de control y Pipelines de CI/CD	Licencia o costo	Documentación y soporte
ACID	No se menciona	Código abierto	No se menciona
BARREL	No se menciona	Código abierto	Soporte y documentación
Checkov	Sí	Código abierto y gratuita	Soporte y documentación
DeepIaC	No se menciona	Código abierto	Soporte y documentación

Foodcritic	No se menciona	Código abierto	Soporte y documentación
GLITCH	Sí	Código abierto	Soporte y documentación
Hayha	No se menciona	Código abierto	Soporte y documentación
KICS	Sí	Código abierto y gratuita	Soporte y documentación
Puppeteer	Solo GitHub y CI/CD	Código abierto	Soporte y documentación
RADON	Solo GitHub y CI/CD	Código abierto	Soporte y documentación
Rehearsal	No se menciona	Código abierto	Soporte y documentación
SecureCode	Solo CI/CD	No se menciona	No se menciona
SLAC	No se menciona	Código abierto	No se menciona
SLIC	No se menciona	Código abierto	Soporte y documentación
SNYK IaC	Sí	Freemium	Soporte y documentación
Sommelier	No se menciona	Código abierto y gratuita	Soporte y documentación
SonarQube	Sí	Freemium	Soporte y documentación
TAMA	No se menciona	Código abierto	Soporte y documentación
Terrascan	Sí	Código abierto y gratuita	Soporte y documentación
Tfsec	Sí	Código abierto y gratuita	Soporte y documentación
Trivy	Sí	Código abierto y gratuita	Soporte y documentación

Fuente: *Elaboración propia*

- **Proceso de Análisis**

El proceso de análisis evalúa la integración con IDEs, sistemas de control de versiones y pipelines de CI/CD, el tipo de licencia y la disponibilidad de documentación y soporte. Se organiza la información en tablas para facilitar la comprensión de las capacidades y limitaciones de cada herramienta.

- **Interpretación Crítica**
  - **Integración completa:** Herramientas como Checkov, KICS, Snyk IaC, SonarQube, Terrascan, Tfsec y Trivy se destacan por su integración completa con IDEs, sistemas de control de versiones y pipelines de CI/CD. Esta integración facilita su adopción en flujos de trabajo DevOps, proporcionando análisis continuos y retroalimentación en tiempo real. Esto mejora la eficiencia en la detección y corrección de problemas de seguridad y calidad del código.
  - **Licencias flexibles:** La mayoría de las herramientas son de código abierto, lo que promueve su uso y contribución dentro de la comunidad de desarrollo de software. Herramientas con licencias Freemium, como Snyk IaC y SonarQube, ofrecen

versiones gratuitas y comerciales, permitiendo a las organizaciones elegir según sus necesidades y presupuesto.

- **Documentación y soporte:** Herramientas como Checkov, KICS, Snyk IaC, SonarQube, Terrascan, Tfsec y Trivy proporcionan documentación detallada y soporte activo, facilitando su implementación y uso efectivo. La comunidad de usuarios y desarrolladores contribuye significativamente a mantener actualizadas estas herramientas, proporcionando soluciones a problemas comunes y mejoras continuas.

2.1.3.4 Preguntas guía para la elección de herramientas de análisis estático, adaptadas a scripts de IaC.

Se presenta un conjunto detallado de preguntas con respecto al criterio de evaluación, la cuales fueron diseñadas para guiar al personal encargado del análisis estático de scripts en IaC en la elección de la herramienta más adecuada según su entorno y requerimientos específicos:

Tabla 13 Preguntas guía para la elección de herramientas de análisis estático, adaptadas a scripts de IaC.

Criterios de evaluación	Pregunta
Compatibilidad con lenguajes y plataformas de IaC:	¿Qué herramienta soporta los lenguajes y plataformas específicos utilizados en los scripts de IaC que se van a analizar?
Modelo de licencia y costo:	¿Qué tipo de licencia requiere su herramienta de análisis estático para scripts de IaC? ¿Existen costos adicionales asociados para acceder a características avanzadas o soporte técnico en su herramienta?
Integración con entornos de desarrollo:	¿Con qué IDEs utilizados por el equipo se debe se integrar la herramienta de análisis estático? ¿Es compatible la herramienta de análisis estático con sistemas de control de versiones como GitHub, GitLab? ¿Puede la herramienta integrarse con los pipelines de CI/CD que usamos en nuestros proyectos de IaC?

Facilidad de Uso y Documentación:	¿La herramienta proporciona documentación clara y detallada, incluyendo guías de inicio rápido, ejemplos prácticos y documentación técnica? ¿Existe un soporte comunitario activo o foros donde se puedan resolver dudas y compartir experiencias?
Capacidades de Análisis y Reporte:	¿Cuál es el objetivo principal de análisis, que realiza la herramienta de análisis estático? ¿Cómo presenta la herramienta los resultados del análisis?
Soporte y Actualizaciones:	¿Con qué frecuencia se actualiza la herramienta para incluir nuevas funcionalidades y corregir errores? ¿El proveedor o la comunidad ofrece soporte técnico en caso de problemas o consultas?
Adaptabilidad y Personalización:	¿La herramienta permita la creación de políticas personalizadas y reglas específicas para el proyecto? ¿Es posible ajustar la herramienta para que se adapte a las necesidades y flujos de trabajo específicos del equipo?
Rendimiento y Escalabilidad:	¿Cómo se comporta la herramienta en términos de rendimiento al analizar grandes volúmenes de código?

***Fuente*:** Elaboración propia

- **Interpretación Crítica**

Al responder estas preguntas, se podrá establecer una base sólida para seleccionar herramientas de análisis estático, que mejor se adapten a los scripts de IaC utilizados en un proyecto en específico. Esta guía proporciona una evaluación inicial y objetiva, facilitando la identificación de herramientas que puedan optimizar la calidad y seguridad del código, así como la eficiencia del equipo de DevOps. Sin embargo, es importante considerar otros aspectos específicos propios de cada proyecto que puedan influir en la elección final.

2.1.4 Conclusiones y Recomendaciones

Se realizó una revisión sistemática y comparativa de las herramientas de análisis estático para scripts de Infraestructura como Código (IaC). Se identificaron 21 herramientas, de las cuales 14 son académicas y 7 populares en la industria, destacando Checkov, KICS, Snyk IaC, Terrascan, TFsec y Trivy por su capacidad para detectar errores de configuración y vulnerabilidades de seguridad en scripts de IaC. Las herramientas fueron clasificadas según

sus características técnicas y operacionales, facilitando así la selección adecuada para diferentes entornos y necesidades.

Los hallazgos de la investigación responden a la pregunta principal al identificar y clasificar las herramientas de análisis estático más utilizadas, proporcionando una guía de preguntas específica para la selección de la herramienta adecuada según las características técnicas y operacionales requeridas por los equipos de DevOps.

Los resultados tienen importantes implicaciones para el campo de DevOps y la gestión de IaC. Las herramientas identificadas no solo ayudan a mejorar la calidad y seguridad del código, sino que también optimizan la eficiencia operativa y la productividad del equipo de desarrollo. Este análisis contribuye significativamente al conocimiento existente, proporcionando una base sólida para futuras investigaciones y desarrollos en el análisis estático de scripts de IaC.

Este estudio se limitó a una revisión documental y no incluyó pruebas prácticas de las herramientas. Además, las herramientas se evaluaron en función de la información disponible en la literatura académica, así como en revistas tecnológicas de renombre, como las publicadas por el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE).

Se sugiere que futuras investigaciones realicen pruebas empíricas de las herramientas de análisis estático en entornos DevOps reales. Además, se recomienda explorar el desarrollo de nuevas técnicas de detección utilizando inteligencia artificial y combinar el análisis estático con otras prácticas de seguridad para mejorar la resiliencia y seguridad de la infraestructura IaC.

Este estudio ofrece una contribución valiosa al conocimiento existente sobre el análisis estático de scripts de IaC, proporcionando una guía práctica para la selección de herramientas adecuadas. Al abordar tanto las características técnicas como operacionales, se ofrece una visión integral que ayuda a los profesionales a tomar decisiones informadas, mejorando así la calidad y seguridad de las infraestructuras gestionadas mediante IaC.

Bibliografía

Aqua Security. (25 de Abril de 2024). *Overview - Trivy*. https://aquasecurity.github.io/trivy/v0.51/docs/

Aquasecurity. (28 de Abril de 2024). *GitHub - aquasecurity/tfsec: Security scanner for your Terraform code*. https://github.com/aquasecurity/tfsec

Atlassian. (29 de Abril de 2024). *What is DevOps?* https://www.atlassian.com/devops

Baran, G. (28 de Abril de 2024). *5 Best Tools to Scan Infrastructure as Code for Vulnerabilities in 2024.* Cyber Security News: https://cybersecuritynews.com/infrastructure-as-code/

Bhalia, V., & Thacker, R. (10 de 09 de 2020). *Mulesoft Code Review Automation Using SonarQube.* DZone: https://dzone.com/articles/mulesoft-code-review-automation-using-sonarqube

Borovits, N., Kumara, I., Krishnan, P., Palma, S. D., Di Nucci, D., Palomba, F., . . . van den Heuvel, W.-J. (2020). DeepIAC: Deep learning-based linguistic anti-pattern detection in IaC. *Proceedings of the 4th ACM SIGSOFT International Workshop on Machine-Learning Techniques for Software-Quality Evaluation (MaLTeSQuE 2020).* New York, NY, USA. https://doi.org/10.1145/3416505.3423564

BridgeCrew. (29 de Abril de 2024). *Checkov*. https://www.checkov.io/

Brogi, A., Canciani, A., & Soldani, J. (2015). *Modelling and analysing cloud application management.* https://doi.org/10.1007/978-3-319-24072-5_2

Brogi, A., Di Tommaso, A., & Soldani, J. (2018). Sommelier: A tool for validating tosca application topologies. En A. Brogi, A. Di Tommaso, & J. Soldani, *Sommelier: A tool for validating tosca application topologies* (págs. 1–22). Springer International Publishing.

Chef. (29 de Abril de 2024). *Cookstyle*. https://github.com/chef/

Chiari, M., De Pascalis, M., & Pradella, M. (2022). Static analysis of infrastructure as code: a survey. *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C).* https://doi.org/10.1109/ICSA-C54293.2022.00049

Dai, T., Karve, A., Koper, G., & Zeng, S. (2020). Automatically detecting risky scripts in infrastructure code. *Proceedings of the 11th ACM Symposium on Cloud Computing.* New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3419111.3421303

Dalla Palma, S., Di Nucci, D., Palomba, F., & Tamburri, D. A. (2022). Withinproject defect prediction of infrastructure-as-code using product and process metrics. *IEEE Transactions on Software Engineering, 48*(6), 2086–2104. https://doi.org/ 10.1109/TSE.2021.3051492

Debois, P. (2009). DevOpsDays Ghent. *DevOpsDays Ghent.* http://www.devopsdays.org/events/2009-ghent/

Feldmann, A., Gasser, O., Lichtblau, F., Pujol, E., Poese, I., Dietzel, C., . . . Smaragdakis, G. (2020). The lockdown effect: Implications of the covid-19 pandemic on internet traffic. *Proceedings of the ACM Internet Measurement Conference.* New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3419394.3423658

Guerriero, M., Garriga, M., Tamburri, D. A., & Palomba, F. (2019). Adoption, support, and challenges of infrastructure-as-code: Insights from industry. *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME).* https://doi.org/10.1109/ICSME.2019.00091

Han, X., Tahir, A., & liang, P. (2021). Understanding Code Smell Detection via Code Review: A Study of the OpenStack Community. *29th IEEE/ACM International Conference on Program Comprehension.* https://doi.org/10.1109/ICPC52881.2021.00038

Hassan, M. M., & Rahman, A. (2022). As code testing: Characterizing test quality in open source ansible development. *2022 IEEE Conference on Software Testing, Verification and Validation (ICST).* Valencia, Spain. https://doi.org/10.1109/ICST53961.2022.00031

Jayaraman, K., Bjørner, N., Outhred, G., & Kaufman, C. (2014). *Automated analysis and debugging of network connectivity policies.*

Jiang, Y., & Adams, B. (2015). Co-evolution of infrastructure and source code: An empirical study. *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories.* https://doi.org/10.1109/MSR.2015.12

KICS. (29 de Abril de 2024). *Kics.io*. https://docs.kics.io/latest/

Kumara, I., Vasileiou, Z., Meditskos, G., Tamburri, D., Heuvel, W.-J. v., Karakostas, A., . . . Kompatsiaris, I. (2020). Towards semantic detection of smells in cloud infrastructure code.

Lepiller, J., Piskac, R., Schaf, M., & Santolucito, M. (2021). Analyzing infrastructure as code to prevent intra-update sniping vulnerabilities. *Tools and Algorithms for the Construction and Analysis of Systems.* Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-72013-1_6

May, C. J. (2023). *Best Practices for Scanning and Securing Infrastructure as Code (IaC)*. https://blog.gitguardian.com/infrastructure-as-code-security-best-practices-cheat-sheet-included/

Morris, K. (2016). *Infrastructure as Code.* O'Reilly Media, Inc.

Rahman, A., Farhana, E., Parnin, C., & Williams, L. (2020). Gang of eight: A defect taxonomy for infrastructure as code scripts. *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE).* https://doi.org/10.1145/3377811.3380409

Rahman, A., Parnin, C., & Williams, L. (2019). The seven sins: Security smells in infrastructure as code scripts. *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE).* https://doi.org/10.1109/icse.2019.00033

Rahman, A., Rahman, M. R., Parnin, C., & Williams, L. (2021). Security smells in ansible and chef scripts: A replication study. *ACM Transactions on Software Engineering and Methodology*(1). https://doi.org/10.1145/3408897

Runterrascan.io. (2022). *Documentation*. https://runterrascan.io/docs/

Saavedra, N., & Ferreira, J. F. (2022). *Glitch: Automated polyglot security smell detection in infrastructure as code*. https://arxiv.org/abs/2205.14371

Schwarz, J., Steffens, A., & Lichter, H. (2018). Code smells in infrastructure as code. *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC).* https://doi.org/10.1109/QUATIC.2018.00040

Semgrep. (29 de Abril de 2024). *Semgrep Make Shift Left Work*. https://semgrep.dev/

Shambaugh, R., Weiss, A., & Guha, A. (2016). Rehearsal: A configuration verification tool for puppet. *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation.* New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2908080.2908083

Sharma, T., Fragkoulis, M., & Spinellis, D. (2016). Does your configuration code smell? *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR).* Austin, Texas. https://doi.org/10.1145/2901739.2901761

Snyk. (29 de Abril de 2024). *Snyk IaC | Snyk User Docs*. https://docs.snyk.io/scan-with-snyk/snyk-iac

SonarSource. (28 de Abril de 2024). *SonarQube 10.5*. https://docs.sonarsource.com/sonarqube/latest/

Swartout, P. (2012). *Continuous Delivery and DevOps: A QuickStart Guide.* Packt Publishing.

Walker, A., & Cerny, T. (2020). Automated Code-Smell Detection in Microservices Through Static Analysis: A Case Study. *Applied Sciences*.

Wang, R. (2022). *Infrastructure as Code, Patterns and Practices: With Examples in Python and Terraform.* Manning Publications.

Willis, J. (30 de Marzo de 2012). *The Convergence of DevOps*. IT Revolution. : https://itrevolution.com/articles/the-convergence-of-devops/

ANEXOS

ANEXO A Lenguajes soportados

- ACID soporta scripts de Puppet para gestión de configuración y automatización. Analiza commits en repositorios OSS, identificando defectos mediante análisis estático y dependencias. Detecta defectos en la configuración y operación.
- Barrel utiliza HTML5 y JavaScript para su interfaz y *backend*. Permite editar y analizar protocolos de gestión en aplicaciones TOSCA de manera interactiva y visual a través de un navegador web.
- Checkov soporta Terraform, CloudFormation, Kubernetes, Docker, Azure Resource Manager y Serverless Framework. Se aplica en CI/CD como GitHub Actions y GitLab CI, escaneando archivos IaC para detectar configuraciones incorrectas y vulnerabilidades.
- DeepIaC soporta Ansible y GitHub, utilizando aprendizaje profundo para analizar scripts de infraestructura, detectando incoherencias y mejorando la calidad del código.
- Foodcritic analiza cookbooks de Chef escritos en Ruby, utilizando una sintaxis específica de Chef.
- GLITCH convierte scripts de IaC (Ansible, Chef, Puppet) en una representación uniforme para detectar malos olores de seguridad, asegurando detecciones consistentes de vulnerabilidades.
- Häyhä utiliza archivos de configuración de CloudFormation, describiendo el estado deseado de una infraestructura en JSON o YAML. Convierte estas configuraciones en grafos de flujo de datos, evaluando posibles vulnerabilidades.
- KICS analiza archivos de configuración de Terraform y Ansible, detectando configuraciones inseguras y problemas de cumplimiento mediante escaneo automático.
- Puppeteer identifica "olores" de configuración en código Puppet mediante análisis estático, mejorando la mantenibilidad y calidad del código sin ejecutarlo.
- RADON Framework predice defectos en scripts de Ansible basados en YAML, utilizando métricas de producto y proceso, y modelos de aprendizaje automático como Random Forest.

- Rehearsal soporta configuraciones de Puppet, transformándolas en grafos de recursos y modelándolos como programas en FS (File System) para verificar consistencia y predictibilidad con un solver SMT.
- SecureCode soporta Shell y PowerShell, detectando patrones de riesgo en playbooks de Ansible. Utiliza ShellCheck y PSScriptAnalyzer para mejorar seguridad, rendimiento y confiabilidad
- SLAC soporta Ansible (YAML) y Chef (Ruby), identificando "security smells" mediante un motor de reglas basado en patrones predefinidos.
- SLIC soporta Puppet, analizando scripts para identificar "security smells" mediante extracción de tokens y aplicación de reglas específicas.
- Snyk IaC soporta Terraform, CloudFormation, Kubernetes, Docker, AWS, Azure y Google Cloud, escaneando configuraciones de IaC para detectar vulnerabilidades y proporcionar soluciones en tiempo real.
- Sommelier soporta TOSCA, validando topologías de aplicaciones en la nube, convirtiendo archivos CSAR o .tosca en representaciones Python para verificar condiciones de interconexión.
- SonarQube soporta lenguajes como Java, JavaScript, Python, C#, C++, y Go, y plataformas como Kubernetes y Terraform. Identifica errores, vulnerabilidades y code smells mediante analizadores específicos.
- TAMA soporta scripts YAML de Ansible, analizando scripts de prueba en proyectos de código abierto en GitHub, detectando "assertion roulette" y "local only testing".
- Terrascan analiza archivos de Terraform, Kubernetes y otros, aplicando políticas definidas en Rego para identificar configuraciones inseguras.
- Tfsec analiza archivos Terraform escritos en HCL, evaluando configuraciones para detectar vulnerabilidades y malas prácticas.
- Trivy analiza archivos de configuración de Kubernetes, Docker y Terraform, detectando vulnerabilidades y errores de configuración. Escanea dependencias en archivos de manifiesto para lenguajes como Go, Java y Python.

ANEXO B Técnica de análisis

Expresiones Regulares:

- (Rahman et al., The seven sins: Security smells in infrastructure as code scripts, 2019) desarrollaron las herramientas SLAC y SLIC, que utilizan expresiones regulares para detectar olores de seguridad en scripts de Ansible y Chef. Estas herramientas fueron extendidas con la creación de ACID, que examina el historial de commits de git para identificar patrones que puedan indicar olores de seguridad en los scripts de IaC.
- (Saavedra y Ferreira, 2022) desarrollaron la herramienta GLITCH, que convierte scripts escritos en Ansible, Chef y Puppet en una forma intermedia representada por un Árbol de Sintaxis Abstracta (AST) y utiliza expresiones regulares para detectar olores de seguridad. Compararon GLITCH con SLAC y SLIC y encontraron que GLITCH tenía una mejor precisión y recuperación.
- Herramientas como Foodcritic (Schwarz et al., 2018) y Puppeteer (Sharma et al., 2016) se utilizan para identificar olores de código en scripts de Chef y Puppet, detectando problemas como la alineación incorrecta del código y declaraciones largas. Cookstyle (Chef, 2024) es una herramienta de linting que ayuda a escribir mejores libros de recetas de Chef corrigiendo automáticamente problemas de estilo, sintaxis y lógica.
- SecureCode detecta riesgos de seguridad causados por la inclusión de scripts de shell como Bash y PowerShell en scripts de gestión de configuración. La herramienta se integra con aplicaciones de terceros como Shellcheck y PSScriptAnalyzer en la canalización CI/CD (Jayaraman et al., 2014).
- Herramientas comerciales como KICS y Semgrep también utilizan expresiones regulares para identificar olores de seguridad en diferentes plataformas de IaC. Checkov y Tfsec se especializan en detectar olores de seguridad en scripts de aprovisionamiento de infraestructura en la nube como Terraform, escaneando scripts de configuración y archivos de paquete en busca de Vulnerabilidades y Exposiciones Comunes (CVE). SonarQube [27] se utiliza para verificar olores de código y seguridad en software de Provisión de Infraestructura de Hardware y Construcción de Imágenes, proporcionando complementos para IDEs para mejorar la calidad del código en desarrollo.

Aprendizaje Profundo y Motor de IA:

- (Borovits et al., 2020) desarrollaron DeepIaC, una herramienta que utiliza una Red Neuronal Convolucional (CNN) para detectar anti-patrones en scripts de Ansible. Utilizaron un conjunto de datos minimalista complementado con errores artificiales para entrenar su modelo, logrando una precisión de detección del 79-92%.
- SNYK IaC es un producto que incorpora Inteligencia Artificial para ayudar a los desarrolladores a identificar olores de código en IaC y proporcionar soluciones para corregirlos. Aunque es una herramienta propietaria y la documentación es limitada, se destaca su uso de técnicas avanzadas de IA para mejorar la calidad del código IaC (Snyk, 2024).

Análisis de Gráficos:

- Brogi et al. [9] desarrollaron BARREL, una herramienta que utiliza análisis de gráficos en plantillas de nodos de TOSCA para verificar la precisión de los planes de gestión. Representar las plantillas como un gráfico ayuda a comprender mejor la infraestructura y verificar la validez del plan creando una representación del estado de la infraestructura.
- Lepiller et al. [16] desarrollaron Häyhä, una herramienta que convierte plantillas o scripts de CloudFormation en gráficos de flujo de datos para detectar vulnerabilidades de francotirador intra-actualización, visualizando estas vulnerabilidades en una interfaz gráfica de usuario (GUI).

Solver SMT (Teoría de Satisfacibilidad de Módulos):

- Shambaugh et al. [19] desarrollaron Rehearsal, una herramienta que utiliza el solver SMT Z3 para identificar olores de código en scripts de Puppet que violan los principios de determinismo e idempotencia. Convierte los scripts en un lenguaje formal y los traduce en especificaciones SMT.
- SecGuru, desarrollado por Jayaraman et al. [20], utiliza el solver SMT Z3 para codificar acciones y políticas del firewall en Microsoft Azure, resolviendo problemas para determinar si permitir o bloquear paquetes de red y gestionando políticas de red a gran escala.

Algoritmos Ad hoc:

- Brogi et al. [26] desarrollaron Sommelier, una herramienta que verifica las conexiones entre entidades en scripts de TOSCA para prevenir referencias indefinidas que pueden causar errores durante la implementación.

Algoritmos de Patrones:

- Rahman et al. [28] desarrollaron TAMA, una herramienta que utiliza algoritmos de patrones para identificar y clasificar errores en scripts de Ansible. TAMA clasifica errores en siete categorías: configuración, dependencia, idempotencia, registro, rendimiento, seguridad y estilo, y fue aplicado a un conjunto de datos de 4,831 scripts de Ansible.

Minería de Datos:

- Dalla Palma et al. [18] desarrollaron RADON, un marco de predicción de defectos que utiliza técnicas de minería de datos y métricas de software para detectar anti-patrones en scripts de Ansible. RADON procesa 108 características relacionadas con el código IaC y utiliza algoritmos como el Bosque de Aislamiento para lograr una precisión del 0.86 y un recuerdo del 0.77.

ANEXO C Categorización de la herramienta en las áreas de IaC

- ACID se destacó en la gestión de configuración al identificar defectos en scripts de Puppet, abordando problemas como configuraciones erróneas, dependencias incorrectas y vulnerabilidades de seguridad.
- Barrel se integró con TOSCA, permitiendo modelar y analizar la infraestructura en la nube, asegurando la correcta orquestación y gestión de componentes y servicios.
- Checkov sobresalió en la gestión de configuración y aprovisionamiento de infraestructura, verificando scripts de IaC en busca de configuraciones erróneas y vulnerabilidades.
- DeepIaC analizó scripts de Ansible para detectar incoherencias y anti patrones lingüísticos, utilizando técnicas de aprendizaje profundo para evaluar la coherencia entre nombres y tareas.
- Foodcritic detectó errores y malas prácticas en cookbooks de Chef escritos en Ruby, manteniendo configuraciones precisas y eficientes.
- GLITCH se destacó en la gestión de configuración mediante la transformación de scripts de Ansible, Chef y Puppet en una representación intermedia, permitiendo detectar malos olores de seguridad.
- Häyhä analizó configuraciones en CloudFormation para detectar vulnerabilidades de seguridad durante las actualizaciones, evaluando dependencias y rutas de acceso en los scripts.
- KICS sobresalió en el aprovisionamiento de infraestructura al analizar configuraciones de Terraform y CloudFormation, y en la gestión de configuración al escanear scripts de Ansible, Chef y Puppet.
- Puppet-Lint verificó la adherencia a las mejores prácticas en scripts de Puppet, mientras que Puppeteer detectó olores de diseño y estructura, asegurando configuraciones mantenibles y eficientes.
- El RADON Framework se destacó en la gestión de configuración utilizando aprendizaje automático para analizar scripts de Ansible, prediciendo defectos y asegurando configuraciones precisas.
- Rehearsal se destacó en la gestión de configuración de Puppet, asegurando configuraciones determinísticas e idempotentes, previniendo errores derivados del orden de ejecución y dependencias.

- SecureCode analizó scripts de Ansible para detectar problemas y vulnerabilidades en Shell y PowerShell, mejorando la seguridad y confiabilidad en la configuración de servidores.
- SLAC se destacó en la gestión de configuración, analizando scripts en lenguajes como Ansible y Chef para detectar "security smells" y mantener configuraciones seguras.
- SLIC se destacó en la gestión de configuración al detectar automáticamente "security smells" en scripts de Puppet, mejorando la seguridad y calidad de los entornos configurados.
- Snyk IaC integró análisis de seguridad en tiempo real en los flujos de trabajo de desarrollo, proporcionando retroalimentación inmediata sobre vulnerabilidades y permitiendo correcciones antes del despliegue.
- Sommelier validó topologías TOSCA en el aprovisionamiento de infraestructura, asegurando interconexiones correctas y cumpliendo condiciones predefinidas.
- SonarQube se destacó en la gestión de configuración analizando scripts de IaC como Ansible y Terraform, detectando errores y vulnerabilidades y mejorando la automatización de la configuración.
- TAMA identificó patrones problemáticos en scripts YAML de Ansible, mejorando la fiabilidad y seguridad de las configuraciones.
- Terrascan evaluó scripts de IaC como Terraform para identificar violaciones de seguridad y cumplimiento, garantizando un aprovisionamiento seguro de la infraestructura.
- Tfsec destacó en el aprovisionamiento de infraestructura mediante la detección de configuraciones inseguras en scripts Terraform, proporcionando recomendaciones antes del despliegue.
- Trivy sobresalió en la gestión de configuración al proporcionar un análisis detallado de configuraciones de IaC, identificando vulnerabilidades y errores de configuración en Terraform, Kubernetes y Docker.

ANEXO D Integración con IDE, control de versiones o CI/CD

ACID se diseñó para identificar defectos en scripts de Puppet mediante análisis estático, sin detallar integraciones con IDE, control de versiones o CI/CD.

Barrel se integró con Open TOSCA para procesar CSARs, sin mencionar integración con IDE, control de versiones o CI/CD.

Checkov se integró con IDEs como Visual Studio Code y sistemas de control de versiones como GitHub y GitLab, facilitando el análisis continuo en CI/CD.

DeepIaC analizó scripts de Ansible con técnicas de aprendizaje profundo, sin integraciones con IDE, control de versiones o CI/CD.

Foodcritic detectó olores de código en cookbooks de Chef, sin detallar integraciones específicas pero adaptable mediante scripts personalizados.

GLITCH se integró con sistemas de control de versiones y CI/CD, permitiendo la adaptación a diversas herramientas y entornos de desarrollo.

Häyhä analizó configuraciones de CloudFormation para detectar vulnerabilidades, sin detallar compatibilidad con otras plataformas.

KICS se integró con IDEs como Visual Studio Code y plataformas CI/CD como Jenkins y GitLab CI para automatizar la seguridad.

Puppeteer analizó repositorios en GitHub y se integró en CI/CD para verificar la calidad del código de configuración de Ansible.

Rehearsal verificó configuraciones de Puppet mediante análisis estático, sugiriendo integración indirecta con flujos de trabajo existentes.

SecureCode se integró con Travis CI para detectar problemas en repositorios de GitHub, facilitando el análisis continuo y la gestión de riesgos.

SLAC detectó "security smells" en scripts de Ansible y Chef, proporcionando resultados en CSV sin detallar integraciones con otras herramientas.

SLIC analizó scripts de Puppet para identificar "security smells", sugiriendo integración con otros sistemas mediante scripts personalizados o APIs.

Snyk IaC se integró con IDEs, SCM y CI/CD, proporcionando retroalimentación en tiempo real sobre vulnerabilidades y permitiendo correcciones directas.

Sommelier validó topologías TOSCA, sin integrar directamente con herramientas de desarrollo, control de versiones o plataformas CI/CD.

SonarQube se integró con IDEs como IntelliJ y Eclipse, GitHub y Bitbucket para control de versiones, y Jenkins y Azure DevOps para CI/CD.

TAMA identificó patrones en scripts de Ansible, sin integraciones directas con herramientas de desarrollo, control de versiones o CI/CD.

Terrascan se integró con GitHub Actions para CI/CD, permitiendo escaneos automáticos de código y ofreciendo integración con diversos IDEs.

Tfsec se integró con VSCode, JetBrains y Vim para análisis en desarrollo, y con GitHub, GitLab y Bitbucket para control de versiones y CI/CD.

Trivy se integró con GitHub Actions y GitLab CI para escaneos de seguridad en CI/CD, soportando Jenkins y Visual Studio Code para detección de vulnerabilidades.

ANEXO E Licencia o costo

- Dado que ACID fue desarrollada en el contexto de una investigación académica, es posible que la herramienta sea accesible para fines de investigación o académicos, pero no se proporcionaron detalles adicionales en el artículo.
- Barrel es de uso gratuito como proyecto de código abierto. Los usuarios pueden acceder a la aplicación a través de un navegador web moderno y descargar el código fuente desde su repositorio en GitHub.
- Checkov es una herramienta open-source y gratuita, disponible bajo la licencia Apache 2.0, permitiendo su uso, modificación y distribución sin costo. Esta licencia promueve el uso y la contribución dentro de la comunidad de desarrollo de software.
- DeepIaC es gratuita para uso personal y educativo sin fines comerciales, con requisitos de citación. Para usos comerciales o redistribución, se requiere permiso previo y/o el pago de una tarifa. Está disponible como software de código abierto en el siguiente URL: https://github.com/swc-rwth/InfrastructureAsCodeSmells.
- La herramienta de análisis estático GLITCH es de código abierto y está disponible gratuitamente en GitHub, permitiendo su uso sin incurrir en gastos adicionales.
- Häyhä se encuentra bajo una licencia Creative Commons Attribution 4.0 International (CC BY 4.0), permitiendo su uso, distribución y modificación gratuita con el crédito adecuado a los autores originales y un enlace a la licencia.
- KICS es una herramienta completamente de código abierto y gratuita, permitiendo a los desarrolladores utilizar y contribuir libremente a su desarrollo.
- Puppeteer es una herramienta de código abierto disponible gratuitamente bajo la licencia Apache 2.0, permitiendo su uso, modificación y distribución incluso para fines comerciales.
- RADON Framework for IaC Defect Prediction es una herramienta de código abierto con licencia Creative Commons Attribution 4.0 License, permitiendo su uso, distribución y modificación gratuita con la atribución adecuada a los autores originales.
- La herramienta de análisis estático Rehearsal es de código abierto y se ofrece sin costo, permitiendo copias digitales o impresas para uso personal o en el aula, siempre que no se distribuyan con fines de lucro o ventaja comercial.

- SecureCode, desarrollada por IBM Research, se menciona en un contexto de investigación sin detalles explícitos sobre soporte o documentación, sugiriendo que podría ser una herramienta interna sin recursos de apoyo disponibles públicamente.
- SLIC es un software de código abierto desarrollado para la investigación académica, sin especificar un modelo de licencia o costo.
- Snyk IaC tiene una licencia freemium, ofreciendo una versión gratuita para funcionalidades básicas y versiones pagadas con características avanzadas y soporte.
- Sommelier es de código abierto, disponible sin costo y se distribuye bajo una licencia de software libre, permitiendo el acceso, modificación y distribución del software libremente.
- SonarQube ofrece una versión gratuita y varias ediciones comerciales. La edición Community es gratuita y de código abierto, mientras que las ediciones Developer, Enterprise y Data Center son comerciales y requieren suscripción.
- La herramienta TAMA es de código abierto y está disponible gratuitamente, diseñada para ser utilizada y modificada libremente por la comunidad, facilitando la mejora continua y la colaboración en su desarrollo.
- Terrascan es una herramienta de código abierto y gratuita bajo la licencia Apache 2.0, permitiendo su uso, modificación y distribución de manera libre.
- TFsec se distribuye bajo la licencia MIT, permitiendo su uso, modificación y distribución libremente, fomentando su adopción y contribución por parte de la comunidad de desarrolladores.
- Trivy se distribuye gratuitamente bajo la licencia Apache 2.0, permitiendo su uso, modificación y distribución sin costo, garantizando libertad y flexibilidad para los desarrolladores y usuarios.

ANEXO F Soporte y documentación

- ACID se describe como una herramienta de análisis estático diseñada para identificar defectos en scripts de Puppet. El artículo se enfocó en la metodología y efectividad de ACID para categorizar defectos, sin proporcionar información sobre soporte o documentación específica.
- Barrel cuenta con documentación accesible en su repositorio de GitHub, donde se puede encontrar el código fuente y las instrucciones para su uso. Cuenta con una interfaz web interactiva que guía a los usuarios en la edición y análisis de protocolos de gestión en TOSCA. Se puede encontrar en: https://github.com/di-unipi-socc/barrel.
- Checkov tiene soporte y documentación extensiva, ofreciendo guías de inicio rápido, ejemplos detallados y documentación técnica en su página oficial y repositorio de GitHub. La comunidad de usuarios contribuye activamente con soluciones y actualizaciones. Se puede encontrar en: https://github.com/bridgecrewo/checkov.
- DeepIaC tiene soporte y documentación disponible en GitHub. La documentación cubre la instalación, uso y ejemplos de aplicación, proporcionando guías detalladas y ejemplos prácticos. Se puede encontrar en: https://github.com/SODALITE-EU/defect-prediction/tree/master/ansible.
- Foodcritic cuenta con documentación detallada sobre su uso e instalación. También ofrece soporte a través de la comunidad de usuarios y desarrolladores, quienes contribuyen con guías y resolución de problemas en foros y repositorios. Se puede encontrar en: https://github.com/Foodcritic/foodcritic.
- GLITCH proporciona soporte y documentación detallada en su repositorio de GitHub, incluyendo instrucciones de instalación, guías de uso y ejemplos prácticos. Se puede encontrar en: https://github.com/sr-lab/GLITCH.
- Häyhä incluye documentación detallada que guía a los usuarios en su uso y configuración, disponible en un repositorio público en GitLab. Se puede encontrar en: https://gitlab.com/rose-yale/hayha.
- KICS tiene soporte y documentación extensiva, incluyendo guías de instalación, integración y uso detallado, además de opciones de contribución y soporte comunitario en GitHub y foros. Se puede encontrar en: https://checkmarx.com/product/opensource/kics-open-source-infrastructure-as-code-project/.

- Puppeteer ofrece documentación accesible en GitHub, incluyendo instrucciones de instalación y ejemplos de uso. Se puede encontrar en: https://pptr.dev/.
- RADON Framework for IaC Defect Prediction tiene soporte y documentación disponible en GitHub y PyPI, proporcionando guías detalladas para su instalación y uso, además de soporte comunitario. Se encontrar en: https://pypi.org/project/repositories-collector/.
- Rehearsal presenta soporte y documentación en línea, ofreciendo el código fuente junto con scripts de evaluación y un apéndice técnico. Se puede encontrar en: https://github.com/plasma-umass/Rehearsal.
- SecureCode, desarrollada por IBM Research, no menciona soporte o documentación, sugiriendo que es una herramienta interna.
- SLAC no ofrece soporte formal ni documentación extensa. Se desarrolla en un contexto académico y se distribuye con el código fuente y datasets disponibles en línea. Se puede encontrar en: https://github.com/rayhanur-rahman/SLIC-Ansible.
- Snyk IaC ofrece soporte y documentación extensiva, incluyendo guías de usuario, ejemplos prácticos y documentación de la API. Se encontra en: https://docs.snyk.io/.
- Sommelier tiene soporte y documentación disponibles en su repositorio de GitHub, incluyendo instrucciones de instalación, uso y ejemplos. Se puede encontrar en: https://github.com/di-unipi-socc/Sommelier.
- SonarQube soporta IaC con analizadores específicos para lenguajes como CloudFormation, Kubernetes y Terraform, proporcionando documentación detallada. Se puede encontrar en: https://github.com/SonarSource/sonar-iac.
- TAMA tiene soporte y documentación disponibles, proporcionando guías detalladas sobre su uso y configuración. Se puede encontrar en: https://github.com/akondrahman/IaCTesting.
- Terrascan tiene soporte y documentación extensa, proporcionando guías detalladas de uso, instalación y configuración. Se encontra en: https://www.tenable.com/terrascan.
- TFsec cuenta con soporte y documentación extensa, incluyendo guías de instalación, uso y configuración. Está disponible en el repositorio oficial de GitHub y el sitio de documentación de Aqua Security. Se puede encontrar en: https://aquasecurity.github.io/tfsec/latest/.
- Trivy cuenta con amplio soporte y documentación detallada, incluyendo guías de instalación, configuración, ejemplos de uso y resolución de problemas. Se puede encontrar en: https://aquasecurity.github.io/trivy/v0.51/.