DevSecOps: Implementación de seguridad en DevOps a través de herramientas open source

Jhon Edison Castro Sánchez
Universitat Oberta de Catalunya
Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones. (MISTIC)
Trabajo de fin de máster. Seguridad empresarial

Consultor: Jordi Guijarro
Profesor responsable de la asignatura: Víctor Garcia Font

Fecha Entrega: 2020/12/29

Agradecimientos

Quiero agradecer a mis padres y mi esposa por el apoyo y cariño constante para la consecución de este logro, su soporte en cada momento hizo el proceso más fácil. En segundo lugar, a mis compañeros y amigos que siempre estuvieron vigilantes del proceso. En último lugar, pero no menos importante al tutor de este trabajo Jordi Guijarro quien desde el ejemplo logró ofrecerme una guía y colaboración constante en la realización del presente trabajo de grado.

FICHA DEL TRABAJO FINAL

Título del trabajo:	*DevSecOps: Implementación de seguridad en DevOps a través de herramientas open source*
Nombre del autor:	*Jhon Edison Castro Sánchez*
Nombre del consultor/a:	*Jordi Guijarro Olivares*
Nombre del PRA:	*Víctor García Font*
Fecha de entrega (mm/aaaa):	12/2020
Titulación:	*Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones*
Área del Trabajo Final:	*Seguridad empresarial*
Idioma del trabajo:	***Castellano***
Palabras clave	*DevSecOps, Integración continua, Open source, gitlab, SAST, DAST, SCA, análisis de secretos, gestión de vulnerabilidades, Automatización, análisis de contenedores, infraestructura como código*

Resumen del Trabajo

En la última década se ha identificado un aumento en popularidad de DevOps como método de desarrollo de software. Este método se muestra como solución a las necesidades de los negocios donde se buscan ciclos de lanzamiento más cortos al igual que grandes cantidades de despliegues de aplicaciones o características hasta cientos de veces a la semana sin afectar la calidad. Sin embargo estos cambios generan la necesidad de evolucionar el entorno de seguridad tradicional hacia uno de mayor rapidez que esté en la misma linea de estas nuevas prácticas sin sacrificar su sentido per se de protección.

Este trabajo presenta algunas herramientas Open Source que sirven como base para la automatización de controles de seguridad en las fases del desarrollo y despliegue del software. Entre estas herramientas están SAST, DAST, SCA, análisis de secretos, el endurecimiento de servidores, la verificación de imágenes de docker y la validación de la infraestructura como código.

En este trabajo se obtiene un sistema operativo (SO) ejecutando Ubuntu 20.04 endurecido, se presenta también un pipeline que cumple con los criterios básicos de seguridad para una aplicación en nodejs.

En esta misma línea se describen otras herramientas a utilizar según el tipo de lenguajes de programación. También se logra identificar que el pipeline creado depende del caso de uso, del lenguaje utilizado y objetivos empresariales. Finalmente, el trabajo futuro se enfoca en la creación de pipelines para aplicaciones serverless, la comparación de herramientas para almacenamiento y gestión de secretos, y la gestión unificada de amenazas y el alertamiento.

Abstract (in English, 250 words or less):

The last decade has seen a rise in popularity of DevOps as a software-development method. This method is shown as a solution to the needs of businesses where shorter release cycles are sought, as well as large numbers of application or feature

deployments, which may be up to hundreds of times a week without affecting quality. However, these changes come with the need to evolve the traditional security environment towards one of greater speed that is in the same line of these new practices without sacrificing its protection.

This work presents some Open-Source tools that serve as a basis for the automation of security checkpoints in the development and launching phases of the software. These tools include SAST, DAST, SCA, secret analysis, server hardening, docker image verification, and infrastructure-as-code validation.

In this job, an operating system (OS) is obtained by running a hardened Ubuntu 20.04 server. A pipeline that meets the basic security criteria for an application in nodejs is also presented.

Along the same line, other tools to be used according to the type of programming languages are described. It is also possible to identify that the use of the pipeline created depends on the use case, the language and the business' goals. Ultimately, the future work focuses on creating pipelines for serverless-applications, comparing tools for secret storage and management, unified threat-management and alerting.

Índice

Lista de tablas

Lista de Ilustraciones

1. Introducción

1.1 Contexto y justificación del Trabajo

Hoy día las empresas para desarrollo de software han optado por cambiar el modelo tradicional, encontrando utilidad en la construcción y entrega de productos rápidos al mercado permitiendo realizar pruebas tanto de concepto como de validación para ideas en corto tiempo (unos pocos días o semanas), lo cual, brinda una oportunidad para la comercialización, minimización del desperdicio, innovación y el diferenciamiento frente a la competencia. Aunado a lo anterior, se genera expansión de productos, visibilidad en el mercado, retorno de inversión mayor. Para poder lograr las características descritas anteriormente, es necesario basar sus modelos en metodologías implementadas a nivel global, es allí, cuando DevOps toma protagonismo, debido a que, aproximadamente el 50% de las organizaciones están empleando esta metodología (Mansfield-Devine, 2018, Stroud, 2020). Una de las características clave y parte importante de su atractivo es que apunta a tiempos de entrega rápidos a través de la automatización de su flujo (Jabbari, Bin Ali, Petersen & Tanveer, 2016).

Hacer que el código esté en producción más frecuentemente hace que los equipos de desarrollo (Dev) y operaciones (Ops), tengan que trabajar de la mano, minimizando los silos, generando que la unión de ambas palabras sean el inicio de la metodología. La mayoría de tareas manuales pasan a ser automatizadas, lo que conlleva a obtener nuevas habilidades transformando las profesiones, especialmente en el área de operaciones. Al incrementar la velocidad de los despliegues los desarrolladores deben tener mayor autonomía y unos objetivos claros, lo que a su vez genera triunfos tempranos, esto ayuda en la felicidad de los desarrolladores lo que se transfiere en una ganancia para el negocio (Gallego, 2020).

Según Carvalho, Moreira & Rosso (s.f.) el enfoque actual de la seguridad es considerado afuera de la fase de desarrollo de una aplicación, primero con ensayos que dependen de la experiencia del equipo, posteriormente mediante pruebas de implantación o hacking ético, para identificar fallos internos o externos lo que siempre ralentiza los procesos y hace que el encontrar problemas sea más costoso, Algunas implementaciones de procesos recientes han optado por incluir la seguridad como parte del ciclo, generando así el concepto de DevSecOps; si bien en la industria el término de DevSecOps es joven, existen diferentes blogs y proveedores que dan orientación sobre el tema, pero cada uno de ellos sugiriendo su producto, es por esto, que una visión académica imparcial ayudará en el proceso de comprender mejor la situación (Koskinen, 2019).

Existen múltiples temas a tratar en la integración de seguridad en ambientes ágiles, uno de ellos es movilizar la seguridad a la izquierda enfocándose en traer un proceso de una fase posterior a etapas tempranas, minimizando defectos productivos. Aunque

la integración de seguridad requiere un enfoque más robusto con énfasis en la colaboración entre equipos, el empoderamiento de las personas para que de forma autónoma se pueda prevenir, detectar y remediar problemas de seguridad.

Para incorporar la seguridad en todas las etapas del desarrollo se debe basar en herramientas que permitan la automatización de los controles, donde se verifique la inexistencia de vulnerabilidades cuando se introduzca un cambio ya sea en la aplicación o en la infraestructura y para esto se pueden utilizar algunas de las mencionadas a continuación:

Herramientas de prueba de seguridad de aplicaciones (AST) que están compuestas por análisis de composición del software (SCA), análisis estático de código (SAST), Análisis dinámico de código (DAST), análisis interactivo de código (IAST), Adicionalmente, herramientas de verificación de secretos en el código, análisis de contenedores, verificación de licenciamiento, especificando estrictamente en el código a desplegar. Si se brinda una visión desde el área de operaciones se hallará una revisión de configuraciones inseguras en infraestructura como código, modelado de amenazas, amenazas de infraestructuras en nube.

Los desarrolladores de software y personal de operaciones que no son expertos en el área de seguridad, pueden, durante un desarrollo, llevarlo involuntariamente hacia software inseguro lo que aumenta el problema del agilismo e incrementa la superficie de ataque (Cappelli, Moore & Trzeciak, 2006).

Las ventajas de DevSecOps están en la velocidad de recuperación frente a desastres, escalabilidad, disminución de la cantidad de amenazas, seguridad desde el diseño, responsabilidad compartida entre los miembros del equipo, mejora iterativa basada en métricas (Raynaud, 2020).

Según Ur Rahman & Williams (2016) el estudio anual realizado por diferentes proveedores del mercado como lo son puppet, circle CI y splunk afirman que el 22% de las compañías con mayores niveles de seguridad tienen un nivel avanzado de prácticas DevOps ya que existe una relación directa los principios de C.A.M.S que se verá en un módulo posterior y los principios que se deben tener en cuenta en seguridad, confiabilidad, predictibilidad, observabilidad y medición.

De hecho, durante la última década, el tiempo promedio desde que se descubre una vulnerabilidad hasta que se repara se ha reducido de 45 a 3 días (Henke, 2020), aunque el estudio de veracode indica que el tiempo medio se encuentra en 171 días, Si se aplica un proceso de DevSecOps el TTR medio para las aplicaciones escaneadas 12 o menos veces al año (menos de una vez al mes, en promedio) es de 68 días. Aquellos con una frecuencia de exploración promedio diaria o más (más de 260 exploraciones) redujeron esa estadística a 19 días. Eso es una reducción del 72% en TTR medio (Jarret, Wysopal & Eng, 2020)

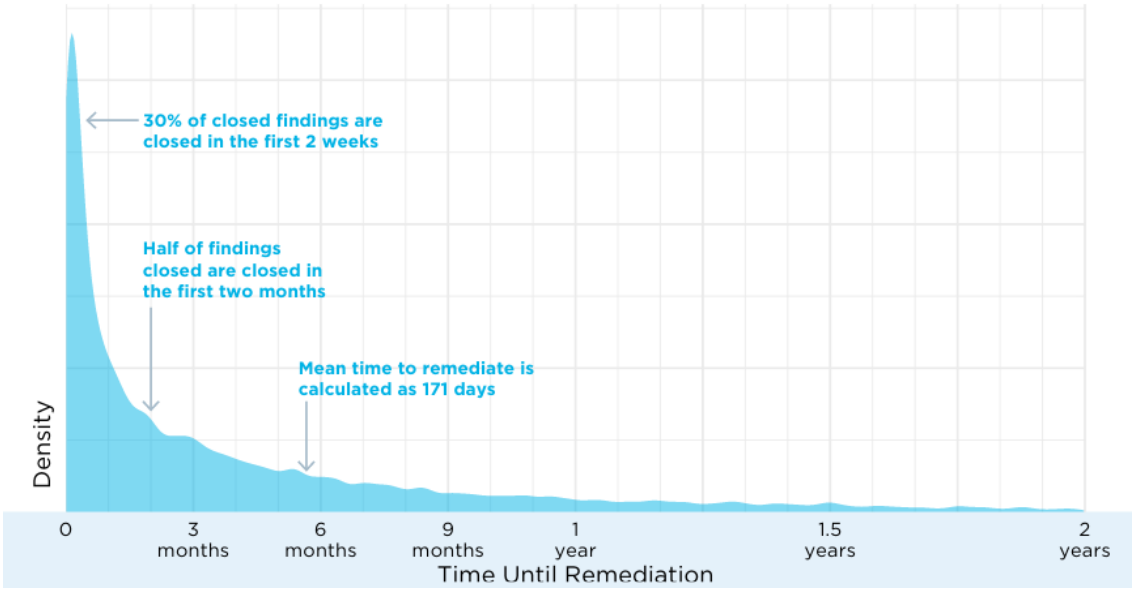

Ilustración 1. Plazos de remediación de falla (Jarret, Wysopal & Eng, 2020)

Se ha evidenciado la correlación presente entre la cantidad de veces que se ejecuta un escaneo y el tiempo para resolver esa vulnerabilidad, es decir son inversamente proporcionales, mientras más escaneos automatizados se ejecuten menos tiempo se tardará el equipo en reparar los defectos encontrados.

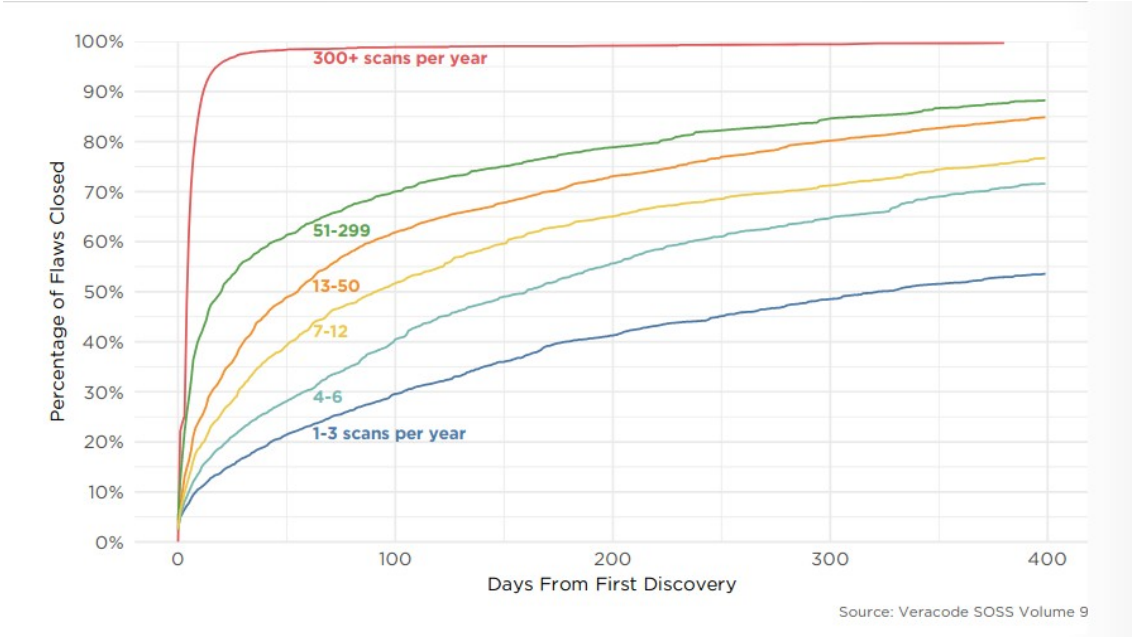

***Ilustración 2**. Día de descubrimiento vs porcentaje de defectos corregidos* (Eng, 2019)

Para Koskinen (2019) el 42% de las actividades de seguridad mencionadas en la investigación de DevOps tienen como objetivo proteger la infraestructura tecnológica, principalmente la infraestructura de contenedores y la nube.

Por esta razón la ejecución del trabajo final de máster se centrará en comprender el estado actual de herramientas open source utilizadas para realizar un movimiento de seguridad a la izquierda en flujos automatizados y de esta manera, poder incluir seguridad desde el diseño en aplicaciones web, teniendo como base el conocimiento del lenguaje de programación usado, framework y la aplicación sobre la que se trabaje, también se pretende, definir los patrones básicos de despliegue en infraestructura como código incluyendo los errores comunes y las buenas prácticas a seguir en ambientes de nube.

Por último, se proporcionan sugerencias para futuras investigaciones sobre la práctica de DevSecOps, para agilizar una forma de medición de seguridad para las implementaciones de DevSecOps.

1.2 Objetivos del Trabajo

1.2.1. Objetivo General:

Implementar dentro del ciclo de vida del desarrollo de aplicaciones, el cómputo automático de herramientas open source que permiten incluir seguridad en el núcleo de la metodología denominada DevOps.

1.2.2. Objetivos Específicos:

1. Analizar las amenazas comunes y frecuentes para las aplicaciones Web, teniendo en cuenta las aplicaciones y la elección de un lenguaje de programación como python, nodejs o php. Estudiando sus aspectos de seguridad y las vulnerabilidades más extendidas.
2. Valorar herramientas open source con base en diferentes métricas para cada uno de los estados, definición de infraestructura, ejecución en ambientes de CI/CD priorizando su evaluación sobre el aporte que ofrece en el incremento de la seguridad en las aplicaciones web.
3. Integrar en la definición de herramientas AST, SCA, detección de secretos, revisión de licencias dentro del ciclo de desarrollo seguro del software como apoyo en el aumento de los estándares de seguridad en las aplicaciones web.
4. Generar un listado de buenas prácticas recolectadas a lo largo del desarrollo del proyecto.

1.3 Problema de Investigación

En mi investigación quiero responder tres preguntas.

1. ¿Cuáles son los errores comunes en seguridad al aplicar infraestructura como código en despliegues automatizados?
2. ¿Cuáles son los desafíos de incluir seguridad en DevOps en Latinoamérica?
3. ¿Cuáles son las herramientas open source que pueden ser utilizadas en cada una de las fases AST, descubrimiento de secretos, SCA?

Para la resolución de las preguntas el trabajo desarrollará:

1. Revisiones automatizadas de la calidad del código de infraestructura generado.
2. Generación de controles automatizados para detección de secretos, SAST, DAST, SCA, revisión de contenedores.
3. Creación de controles de seguridad automáticos de la infraestructura con sus respectivas alertas.
4. Revisión de buenas prácticas trabajadas en cada uno de los controles definidos.

1.3 Enfoque y método seguido

En ingeniería de software el método de investigación MLR que se considera una forma de Revisión de Literatura Sistemática (SLR) con la inclusión de literatura gris ha venido tomando mayor fuerza (Neto, Santos, Endo & Fagundes, 2019, Zhou, 2020), por esta razón en la tesis final se opta por utilizar este método investigativo.

El proceso a seguir consta de una revisión de la literatura utilizando el método MLR definiendo allí la relevancia teórica de cada artículo científico y así poder identificar conceptos previos relacionadas con las preguntas de investigación, después de esto se deben generar pipelines independientes y de esta forma poder cumplir cada objetivo del punto anterior.

1.4 Planificación del Trabajo

En la ilustración 3 se presenta el plan de trabajo de grado y en la 4 su respectivo diagrama de Gantt.

Nombre de la tarea	Fecha de inicio	Fecha de terminación
TFM		
Entrega 1 - Plan de trabajo	**09/16/2020**	**09/29/2020**
Definición de objetivos	09/16/2020	09/24/2020
Planificación	09/25/2020	09/25/2020
Contexto y justificación del trabajo	09/26/2020	09/29/2020
Entrega 2 - Definición de los fundamentos del proyecto	**09/30/2020**	**10/27/2020**
Definición de términos Devops, DevSecOps, CI / CD , AST, SCA, modelado de amenazas, infraestructuras ágiles	09/30/2020	10/09/2020
Definición de cultura y gestión en DevSecOps, modelos de madurez	10/10/2020	10/18/2020
Revisión herramientas open source SCA, AST, verificación de secretos, validación de infraestructura, CI a utilizar	10/19/2020	10/27/2020
Entrega 3 - Implementación de las herramientas elegidas en el pipeline	**10/28/2020**	**11/24/2020**
Puesta en marcha de la herramienta de validación de infraestructura como código	10/28/2020	11/3/2020
Puesta en marcha de la herramienta de análisis de secretos elegida	11/3/2020	11/8/2020
Puesta en marcha de la herramienta de SCA elegida	11/9/2020	11/14/2020
Puesta en marcha de la herramienta de SAST elegida	11/15/2020	11/20/2020
Puesta en marcha de la herrmienta de análisis de container elegida	11/21/2020	11/24/2020
Entrega 4 - Memoria final	**09/16/2020**	**12/29/2020**
Documento final, revisión de estilo	9/16/2020	12/22/2020
Integración de todas las herramientas	11/25/2020	12/15/2020
Conclusiones y trabajo futuro	11/25/2020	12/1/2020
Abstract	12/2/2020	12/10/2020
Revisión memoria y correción de errores	12/26/2020	12/29/2020
Entrega 5 - Presentación del vídeo	**12/30/2020**	**01/05/2021**
Realización diapositiva	1/2/2021	1/3/2021
Realización del vídeo	1/4/2021	1/5/2021
Entrega 6 - Defensa del TFM	**01/11/2021**	**01/15/2021**
Defensa del TFM	1/11/2021	1/15/2021

***Ilustración 3**. Plan de Trabajo de grado*

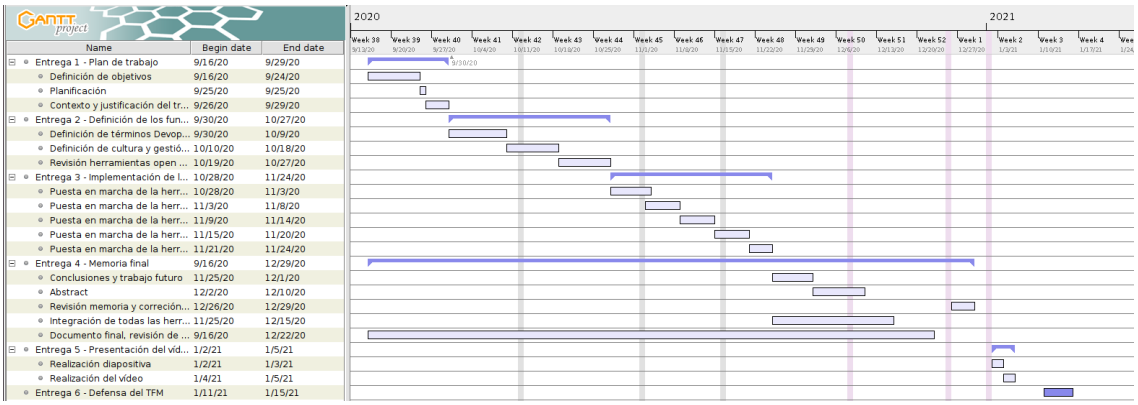

***Ilustración 4**. Diagrama de Gantt del Trabajo de grado*

1.5 Breve sumario de productos obtenidos

- Código en ansible para el endurecimiento de servidores con ubuntu 20.04
- Pipeline que consta de 14 trabajos, tres linters, uno para docker , otro para ansible y otro para terraform. Validación de SAST en infraestructura como código con checkov y tfsec, aplicación de controles de SCA con npm-audit, owasp dependency check y retire.js, detección de secretos con git leaks y TruffleHog, pruebas unitarias, uso de nodejssan y sonar en SAST, escaneo de imágenes docker con trivy.
- Código fuente de la solución establecida tanto para la creación de la infraestructura como para el aseguramiento de la misma.

2. Marco teórico

Este capítulo contiene una descripción general de los antecedentes teóricos necesarios para comprender el enfoque del trabajo final de máster. Se ilustran los conceptos clave de las preguntas de investigación y se presentan las definiciones relevantes para conceptos y metodologías relacionadas. Además, se discute el estado del arte del tema en cuestión y se hace una revisión de literatura previa.

2.1 Conceptos clave

2.1.1 SDLC (Ciclo de vida para desarrollo de software)

El origen de cualquier software se da por algo llamado el ciclo de vida para desarrollo de software. Tradicionalmente, antes de que el software llegue a los computadores, dispositivos móviles, dispositivos IOT entre otros, se somete a un proceso riguroso y complejo. Dado que el software actual, es un producto abstracto, multifuncional que debe respetar los principios básicos de calidad y seguridad, el camino desde la ideación hasta la puesta en producción puede ser bastante largo. Motivo por el cual se define y utiliza un método de desarrollo denominado SDLC constituido por un conjunto de fases. Cada una de estas múltiples fases (de diseño y desarrollo) tienen un conjunto de pasos bien definidos y organizados. Estos, son la definición de requisitos, el diseño del software, la fase de codificación o implementación, la fase de pruebas y por último la entrega del producto. Constituyendo un modelo para desarrollo en cascada donde cada fase, debe ser finalizada para que se pueda iniciar la siguiente, es un modelo tradicional que es sencillo y aplicable a múltiples proyectos para desarrollo de software, pero que presenta un defecto importante en los procesos comerciales actuales donde el tiempo de ir al mercado es fundamental (Balaji & Murugaiyan, 2012).

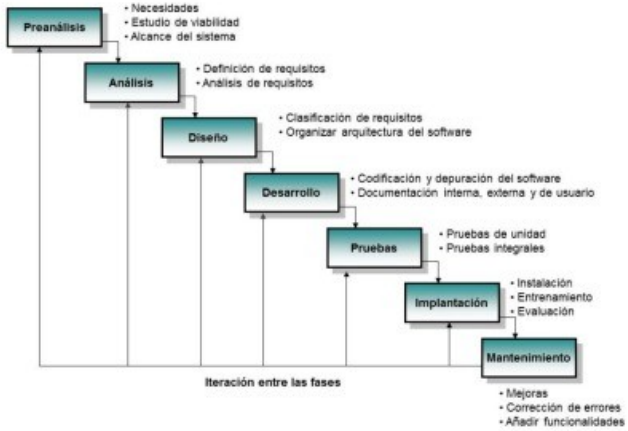

***Ilustración 5**. Ciclo de vida modelo en cascada*

El modelo de cascada ha sido criticado por sus desafíos para adaptarse a los cambios, esto debido a que los proyectos de software a menudo tardan mucho en finalizar, es decir el lapso de tiempo desde la fase de requisitos hasta la fase de implementación puede ser de meses o incluso años lo que lo hace difícil para evolucionar según las necesidades del cliente (Cois, Yankel & Connell, 2014).

En proyectos extensos o en emprendimientos las necesidades del cliente o del mercado pueden cambiar durante el desarrollo del producto, lo que hace que al seguir este tipo de metodologías se pueda entregar un producto de software que no cumple con las expectativas o que está desactualizado.

Al tener entornos cambiantes y de evolución constante la necesidad de una metodología que sea capaz de involucrar al cliente y sus comentarios durante el desarrollo surgió y fue allí donde se creó un modelo de desarrollo ágil.

2.1.2 Metodología de desarrollo ágil

Inicialmente, se consideró que los métodos ágiles sólo eran adecuados para equipos pequeños, pero en los últimos años diferentes autores identificaron la necesidad de escalar el concepto ágil a nivel empresarial (Fitzgerald & Stol, 2015).

En el desarrollo ágil la atención se centra en algo iterativo e incremental. Este tipo de desarrollos están basados en el manifiesto ágil (Beck et al, 2001). Un manifiesto creado por conocedores frustrados de la industria que anhelaban un desarrollo que fuera, como el nombre elegido lo indica, ágil y dinámico. La forma ágil ha ganado popularidad en sus casi 20 años, reemplazando el estilo de desarrollo en cascada casi por completo (Fitzgerald & Stol, 2015). Ágil ofrece adaptabilidad dinámica, ya que, el software se construye en pequeños incrementos que permiten adaptarse a las nuevas necesidades comerciales a medida que surgen, con esto, brinda la capacidad de adaptarse a un mercado cambiante.

Los métodos de gestión de proyectos que potencian los principios ágiles de gestión como XP, SCRUM o Kanban se adoptan en muchas empresas. El concepto de una práctica LEAN, en la que se enfatiza la colaboración, el aprendizaje del fracaso y la retro-alimentación, también está ganando popularidad y fortalece los valores ágiles (Poppendieck & Poppendieck, 2003). El objetivo general del método de desarrollo ágil es una mejora de la eficiencia en el desarrollo y la capacidad de adaptarse a un ritmo rápido marcado por las solicitudes del cliente. Como parte de la metodología, se crearon muchas prácticas operacionales y de desarrollo ajustadas o completamente nuevas. Una técnica importante que es relevante para este estudio y parte inicial del tema en cuestión es la práctica de integración continua y entrega continua.

2.1.3 Integración continua, entrega continua, despliegue continuo.

Integración continua

La actividad operativa de integración y entrega continua (CI / CD) se formó a partir de la adopción de principios ágiles en el desarrollo de software. El factor principal que impulsa el ritmo rápido y la adaptabilidad al mercado es una rápida integración y entrega de software.

La integración continua (CI) es una experiencia para desarrollo que requiere que los programadores integren el código en un depósito distribuido priódicamente al día. Cada registro se verifica luego mediante una compilación automatizada, facilitando a los equipos la identificación de problemas en una etapa temprana (D*uvall, Matyas & Glover, 2007).*

Para validar si realmente se está haciendo integración continua se pueden realizar estas tres preguntas básicas:

1. ¿Cada desarrollador hace commit por lo menos una vez al día en la línea principal compartida?
2. ¿Cada commit dispara una construcción automática y sus pruebas?
3. Si la construcción y las pruebas fallan ¿Esto será reparado en los próximos 10 minutos?

Las anteriores preguntas, llegan al núcleo de la integración continua. La idea es que nadie esté trabajando en una base de código que se desvíe significativamente de la de los demás. La integración continua significa que el equipo sabe cuál es realmente el estado actual del código, evitando grandes fusiones riesgosas para que las personas puedan refactorizar todo lo que necesiten. La opinión común es pensar que al contar con un servidor de integración continua se diverge totalmente de la descripción principal propuesta por Kent Beck, como parte de XP al no tener relación con herramientas, siendo un flujo de trabajo humano, donde las personas se comprometen a alimentarlo todos los días, ejecutando un proceso tecnológico para repositorios de código surgido después y que es muy útil actualmente (Fowler, 2020).

La integración continua se respalda por varias prácticas como lo son:

1. Mantener un repositorio fuente único
2. Automatizar la compilación
3. Hacer que la compilación se auto diagnostique
4. Mantener la construcción rápida
5. Facilitar a cualquiera obtener la última versión ejecutable
6. Todos pueden ver lo que está pasando
7. Automatizar la implementación

Si lo llevamos a un proceso operacional se debería realizar de la siguiente manera:

Los desarrolladores verifican el código en sus espacios de trabajo privados
Cuando haya terminado, el desarrollador hace commit al repositorio
El servidor de CI verifica el repositorio y comprueba los cambios cuando se producen.
El servidor de CI construye el sistema y ejecuta pruebas unitarias y de integración.
El servidor de CI libera artefactos desplegables para pruebas
El servidor de CI asigna una etiqueta de compilación a la versión del código que acaba de compilar.
El servidor de CI informa al equipo la construcción exitosa.
Si la compilación o las pruebas fallan, el servidor de CI alerta al equipo.
El equipo soluciona el problema lo antes posible
El equipo continúa integrando y probando a lo largo del proyecto.

Dentro del equipo existen diferentes responsabilidades que se deben asumir como lo son:

1. Revisión del código con frecuencia
2. No hacer registro de código roto
3. No hacer registro de código no probado
4. No hacer registro cuando la compilación esté rota

Entrega continua:

Esta contribuye a empresas grandes a ser tan diligentes e innovadoras como las nuevas organizaciones, con promociones de confiabilidad y bajo riesgo, la entrega continua posibilita la adaptación continua del software según los reportes de los usuarios, las transformaciones del mercado y las modificaciones en la estrategia comercial (*Humble. & Farley. 2010)*.

La entrega continua es la extensión natural de la integración continua, un enfoque en el que los equipos se aseguran de que cada cambio en el sistema sea liberable, este libera cualquier versión del software con solo presionar un botón de modo que se pueda entregar cambios con frecuencia y obtener comentarios rápidos sobre lo que les importa a los usuarios. Cuando esta se ejecuta correctamente, los programadores contarán con un mecanismo disponible para implementarse que será sometido a unas pruebas estandarizada. “La entrega continua automatiza todo el proceso de publicación de software. Cada revisión efectuada activa un proceso automatizado que crea, prueba y almacena la actualización. La decisión definitiva de implementarla en un entorno de producción en vivo la toma el desarrollador” (¿Qué es la entrega continua? – Amazon Web Services. 2020).

La práctica de CI / CD hace hincapié en un flujo continuo de integración y publicación de código mediante el uso de sistemas automatizados *(Chen, 2015)*

Se deben evitar algunos anti patrones como lo son:

1. Desplegar el software manualmente
2. Despliegues al ambiente productivo solo si el ambiente de desarrollo está completo.
3. Configuraciones manuales de ambientes productivos.
4. No tener definidos sistemas de alertas.

Despliegue continuo:

Constituye el siguiente paso de la entrega continua: toda modificación enviada a pruebas automatizadas se ejecuta de forma mecánica en producción. La implementación continua es el propósito de organizaciones con limitados requisitos normativos o diferentes. Su aspecto fundamental es tomar la decisión continua y pertinente para la organización según sus necesidades, sin limitaciones en TI.

Su discrepancia con la entrega continua es la publicación persistente de actualizaciones a los usuarios, en cambio, dichas actualizaciones son liberadas de forma manual. Al contrario, el despliegue continuo es una acción para desarrollo de software en la que cada cambio de código se envía a producción de forma automática, luego de ir al pipeline, generando varios cambios de envío diario a producción (ver Ilustración 6).

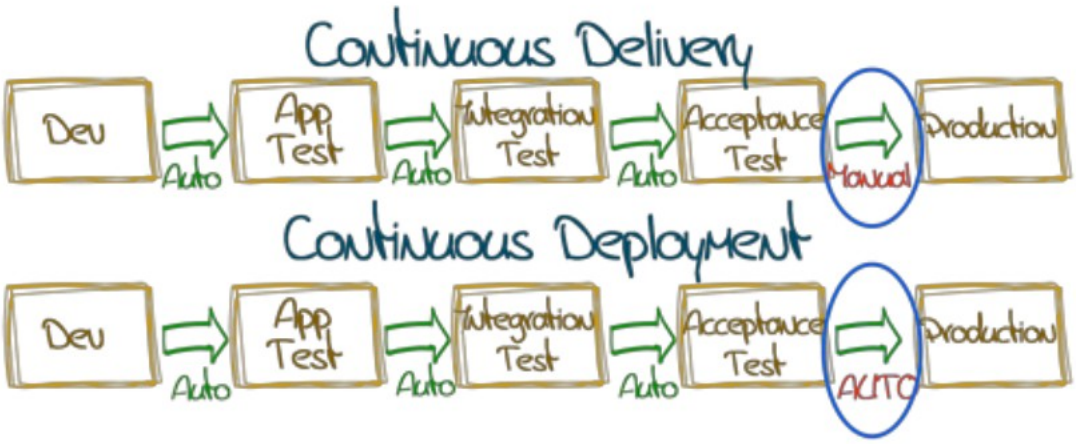

Ilustración 6*. Entrega continua vs despliegue continuo*

Si bien la práctica es flexible y rápida, también es propensa a fallas de comunicación, desalineación entre los equipos de desarrollo y operaciones. La separación de estos equipos es el mayor defecto que actúa principalmente en contra de la flexibilidad y la práctica a prueba de errores (Chen, 2015). Una práctica que contrarresta este defecto y rompe el muro imaginario entre los dos equipos es la metodología DevOps.

Tanto CI como CD forman la columna vertebral del entorno moderno de DevOps y DevSecOps.

2.1.4 DevOps

Existen diferentes lugares que brindan indicio sobre la palabra DevOps, uno de los más concretos nos lleva al año 2008 donde se empieza a emplear el vocablo infraestructura ágil en algunos foros de internet, estos tenían como objetivo las temáticas de desarrollo ágil a nivel de infraestructura inspirada en la ya conocida metodología ágil para desarrollo, los dos involucrados en este diálogo fueron Patrick Debois y Andrew Shafer quienes generaron un hilo de internet para hablar del tema, posteriormente en un foro europeo agile sysadmin de debate se inició con la temática propiamente (Willis, 2012).

El término DevOps fue creado en 2009 durante la Conferencia Velocity da O'Reilly, en esta conferencia John Allspaw (Etsy.com) y Paul Hammond (Typekit) presentaron el trabajo 10+ Deploys Per Day: Dev and Ops Cooperation at Flickr (Carvalho, Moreira & Rosso, s.f.).

El concepto DevOps surgió de la creciente desconexión y los problemas que surgen dentro de las grandes empresas de software entre las áreas de desarrollo y operaciones y la necesidad de alinearlas para realizar la implementación del software en producción, también de la inspiración de Debois por replicar la metodología a nivel mundial logrando con ello crear el evento DevOpsDays (Debois, 2009). A medida que las organizaciones escalan, también lo hacen el desarrollo y la operación, y aunque inicialmente pueden tener vínculos de comunicación estrechos, un mayor tamaño del equipo y una separación más estricta de responsabilidades pueden debilitar dichos vínculos (Swartout, 2012).

La definición académica de DevOps según Jabbari, Bin Ali, Petersen & Tanveer, (2016, p. 9) es: "DevOps es una metodología de desarrollo (C4) destinada a cerrar la brecha (C3) entre Desarrollo (Dev) y Operaciones (C1), enfatizando la comunicación y la colaboración (C2), la integración continua (C7), aseguramiento de la calidad (C8) y entrega (C5) con implementación automatizada (C6) utilizando un conjunto de prácticas de desarrollo".

Otros objetivos de DevOps que están en discusión académica son la entrega de productos fiables (Lee, 2018, p.18), la capacidad de ajustar y satisfacer las necesidades empresariales dinámicas (Mansfield-Devine, 2018), y en última instancia la comercialización del software de forma más rápida (Tamburri, Di Nucci, Di Giacomo & Palomba, 2018).

DevOps es considerada una extensión de ágil, sus métodos ágiles permiten las prácticas de DevOps, tanto DevOps como ágil, responden a la necesidad de diferentes partes interesadas, ya que, DevOps mejora la relación entre desarrollo y operaciones,

mientras que las metodologías ágiles hacen lo mismo con los propietarios del negocio y los desarrolladores (Jabbari, Bin Ali, Petersen & Tanveer, 2016).

Debido a su carácter acelerado, estas prácticas también, en última instancia, “. . . somewhat neglect security . . .” (Swartout, 2012) descuidan un poco la seguridad. Esta gran falta de enfoque a menudo se critica cuando DevOps se analiza en la literatura académica desde un punto de vista centrado en la seguridad Ver Ilustración 7. (Koskinen, 2020).

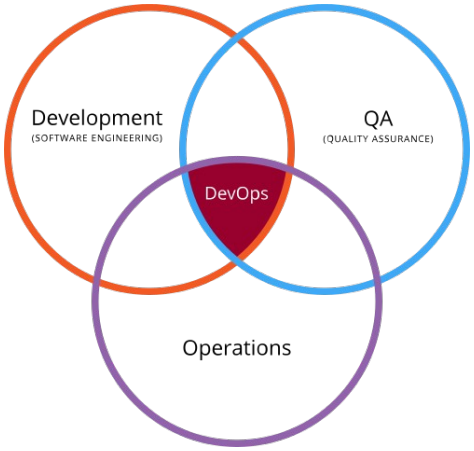

***Ilustración 7**. Intersección DevOps.*

Según los estudios del estado de DevOps (ver Ilustración 8) generados por puppet desde el 2013 y DORA de Google desde el 2018, está demostrado que al utilizar este tipo de metodologías se logran mejoras en 4 aspectos, velocidad de despliegue de hasta 208 veces más frecuentes que organizaciones tradicionales, tiempos de recuperación de fallas 106 veces más rápidas, tasas de fallo ante cambios 7 veces menor y 2604 veces un tiempo más rápido de recuperación de incidentes, enfocadas en dos variables rendimiento y escalabilidad.

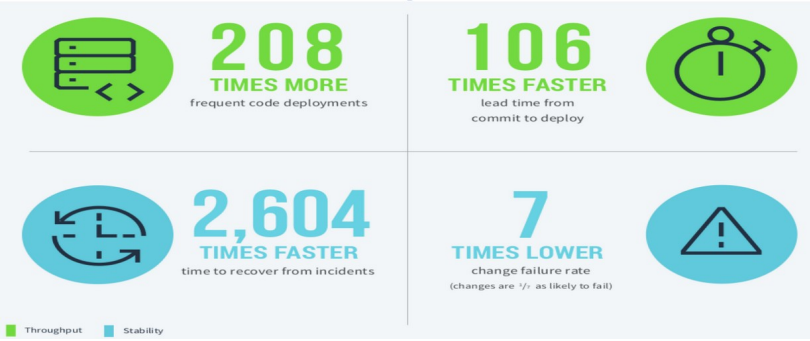

***Ilustración 8**. Estadísticas Informe Puppet 2019.*

Una práctica de DevOps (Ver Ilustración 9) se define en forma de infinito debido a sus continuos pasos recurrentes en el proceso. Atlassian define los pasos principales del proceso de DevOps como planificación, construcción e integración continua en el lado izquierdo seguidos del despliegue, operación y retro-alimentación 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 retro-alimentación continua y la colaboración del lado de operaciones con el lado del desarrollo y viceversa.

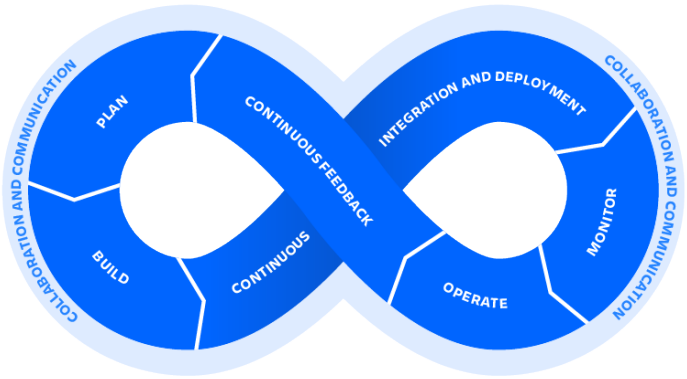

***Ilustración 9**. Ciclo de desarrollo en DevOps atlassian.*

En otras definiciones el ciclo infinito más cercano (ver ilustración 10) a los temas tratados en una metodología en cascada, pero con un desarrollo más colaborativo y continuo, esto se alinea con la mayoría de las otras definiciones del ciclo de vida de desarrollo de DevOps.

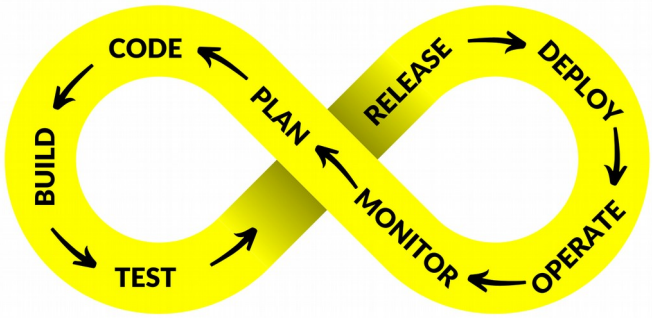

***Ilustración 10**. Ciclo de desarrollo en DevOps* victorops.

2.1.4.1 Los 4 +1 principios de DevOps CA(L)MS

La metodología DevOps se basa en cuatro principios. Cultura, automatización, medición, y compartir que vienen del acrónimo en inglés Culture, Automation, Measurement y sharing. Estos principios fueron introducidos por primera vez en 2010 por los pioneros del campo (Willis, 2010). Desde ese momento han sido usados como un marco teórico tanto en la academia (Myrbakken & Colomo-Palacios, 2017, Humble & Molesky, 2020) como en la industria para presentación de informes sobre el tema (Puppet, 2019, (DevOps Research & Assessment (DORA), 2020).

En la literatura se encuentra un principio adicional, de allí la letra L que viene de aplicar el principio lean, es allí que depende de la literatura consultada podremos obtener CAMS o CALMS (Atlassian, 2020) según sea el caso.

A continuación, se definen los principios con mayor detalle:

Cultura o colaboración:

El primer paso hacia DevOps es romper los silos: En DevOps, un equipo de desarrolladores no hace un producto en un lugar y luego lo transmite a un equipo de operaciones en otro lugar. En DevOps, la responsabilidad se comparte (Gallego, 2019, Cois, Yankel & Connell, 2014). por lo tanto, un defecto o un problema también se convierte en un problema compartido transformándose en un modelo de cooperación y trabajo hacia un objetivo común.

Las herramientas de automatización existentes son inútiles a menos que los profesionales de desarrollo y operaciones trabajen juntos. Porque DevOps no resuelve problemas de herramientas. Resuelve problemas humanos (Atlassian, 2020).

En un estudio de Gartner se predice que para el 2022 el 75% de las iniciativas de DevOps no cumplirán las expectativas debido a problemas relacionados con el aprendizaje y el cambio organizacional (Gartner, 2019).

Primero las personas y el proceso. Si no se tiene cultura, todos los intentos de automatización serán infructuosos (Willis, 2020).

El propósito es migrar de los silos al trabajo conjunto bajo una cultura establecida como se muestra en la ilustración 11.

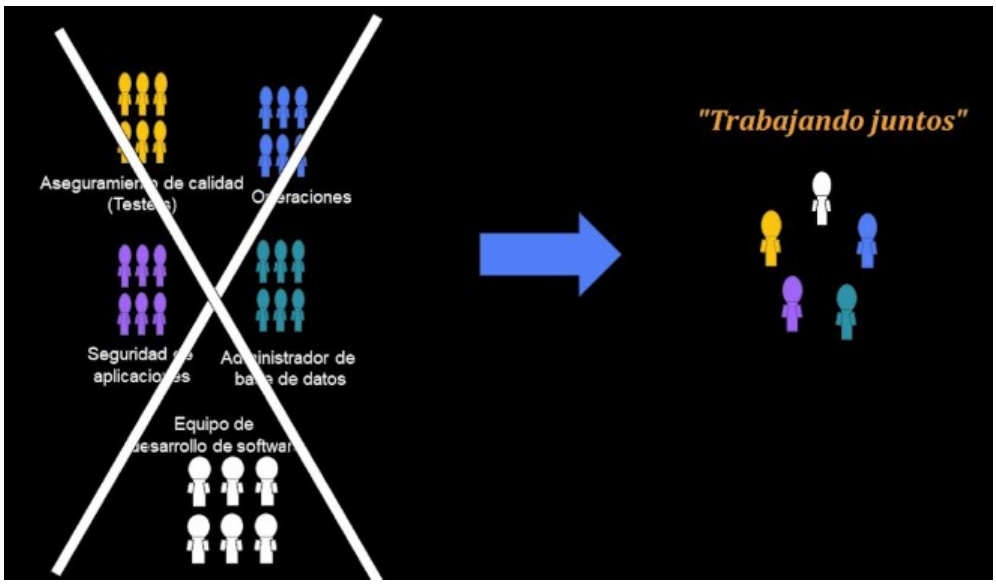

***Ilustración 11**. Transición a DevOps.*

Si la organización sólo adopta las tecnologías DevOps y herramientas automatizadas, surgirán problemas debido a que las personas no están lo suficientemente actualizadas con el cambio cultural que demanda la tecnología.

La adopción de DevOps cambia la cultura organizacional a través del cambio de roles: por ejemplo, si todas las pruebas están automatizadas, los antiguos evaluadores deben adoptar nuevos roles en la organización. Lo que comienza como adopción de una tecnología de implementación más rápida, de hecho, puede convertirse en un cambio de cultura en toda la organización (Bass, Holz, Rimba, Tran & Zhu, 2015).

La cultura orientada a la seguridad no puede nacer sin una clara orientación hacia ella. Sin una dirección clara, el riesgo con DevOps es hacer entregas rápidas sin prestar atención a la seguridad (Koskinen, 2020).

Automatización:

La principal diferencia que distingue a DevOps de otras prácticas es su enfoque en el desarrollo y lanzamiento de software automatizado. Esta práctica se realiza mediante pipelines.

Una de las características más usadas y que quizás llama la atención de la parte técnica son los pipelines de despliegue, allí se hacen las construcciones, obtención de artefactos de software, pruebas, y el despliegue, todo de forma automatizada. J humble define el pipeline de despliegue como “a single path to production for all changes to a given system, whether to code, infrastructure and environments, database schemas and reference data, or configuration.” El proceso de automatización mejora la productividad y el tiempo de comercialización, pero también aumenta la independencia de los desarrolladores individuales o los equipos. Este proceso allana el camino para

múltiples implementaciones al día que permiten que las mejoras o corrección de errores en el software sean publicados sin esperar la aprobación o liberación de otras dependencias.

Existen varias organizaciones utilizando la automatización como base de su negocio, Etsy implementa cambios en sus servidores de producción 50 veces al día con menos interrupciones que cuando la empresa utilizó un enfoque en cascada. Después de mudarse a su propia nube, los ingenieros de Amazon implementan código cada 11,7 segundos, en promedio, lo que reduce tanto la cantidad como la duración de las interrupciones al mismo tiempo. Los ingenieros de Netflix implementan código miles de veces al día (Null, 2020). En Latinoamérica mercado libre hace más de 1500 despliegues por día.

Otro aspecto importante con la automatización es el acceso a retro-alimentación inmediata por parte de los diferentes equipos, desarrollo puede asegurar la calidad del código escrito si las pruebas establecidas para tal fin pasan correctamente, operaciones puede ejecutar herramientas para validar que los despliegues cumplen con las características y seguridad puede revisar en el pipeline si se cumple con los criterios mínimos de seguridad y en caso tal que el software siga su camino a producción.

Existen un conjunto de puntos que debemos automatizar en la metodología DevOps:

1. Nuestro código debe estar en un sistema de control de versiones.
2. Debe existir integración continua, entrega continua, despliegue continuo.
3. El testing debe estar automatizado (Puede existir parte manual)
4. El manejo de configuraciones y despliegues debe ser automatizado.
5. Debe existir monitoreo continuo.

Medición o monitoreo:

El proceso de monitoreo se trata de observar e indicar el comportamiento de los sistemas, a menudo con un motivo oculto de determinar lo que es correcto. Su propósito es responder a la pregunta siempre presente: ¿Mis sistemas están haciendo lo que se supone que deben hacer? este principio en general cae más en las manos del equipo de operaciones y puede verse en parte como un subproducto de los pasos de automatización planteados antes (Schlossnagle, 2018).

El proceso de seguimiento y medición se implementa para poder detectar errores lo antes posible y de esta forma mejorar la calidad de los productos generados, de acá también se obtienen oportunidades para el aprendizaje y crecimiento, para ello se deben tener un conjunto de medidas aplicables al negocio alineadas con los objetivos comerciales. No solamente se debe medir sino utilizar los resultados de las mismas como insumo para formar un circuito de retro-alimentación que impactará en las mejoras del producto

Si no se puede medir, no se puede mejorar. Una implementación exitosa de DevOps medirá todo lo que pueda con tanta frecuencia como sea posible. Métricas de rendimiento, métricas de procesos e incluso métricas de personas (Willis, 2020).

Debe existir un ciclo de retro-alimentación desde operaciones hacia desarrollo para poder proporcionar una mejor comprensión de la seguridad y poder solucionar rápidamente cualquier problema de lógica o de seguridad que surja en el desarrollo (Koskinen, 2020).

Compartir / intercambiar:

La idea es compartir tanto los éxitos como los fracasos con los equipos, departamentos y dentro de la organización. Esta práctica fortalece el aprendizaje de toda la comunidad y va en línea con el objetivo común de desarrollar software de gran calidad en un esfuerzo conjunto (Puppet, 2019, Humble & Molesky, 2020, Moreira & Ibiri, 2020).

Los desarrolladores y los equipos de operaciones describen el problema como el enemigo, no entre ellos. Otra motivación interesante en el movimiento DevOps es la forma en que compartir historias de éxito de DevOps ayuda a otros. Primero, atrae talento, y segundo, existe la creencia de que al exponer ideas se puede crear una gran retro-alimentación (Willis, 2020).

En una organización donde los fracasos no se comparten, cada equipo debe tropezar con los mismos obstáculos. Donde los éxitos no se comparten, las prácticas exitosas se convierten en secretos de los equipos que las inventaron. Para que toda la organización se beneficie y crezca, compartir debe ser un principio sólido que se aplique en toda la organización. Los desarrolladores, operaciones y el área de seguridad deben trabajar juntos para hacer que el producto sea lo mejor posible. Con una mentalidad de compartir, los problemas no se tratan en silos aislados, sino que hay un esfuerzo colaborativo. El comportamiento del producto en producción trae retro-alimentación a los desarrolladores y viceversa (Koskinen, 2020).

El Departamento de Defensa (DoD) sigue utilizando principalmente metodologías de software Waterfall con entrega de software cada 3 a 10 años, lo que hace imposible mantenerse al día con el ritmo de la tecnología, por esta razón decidieron hacer un cambio progresivo a DevOps (ver ilustración 12).

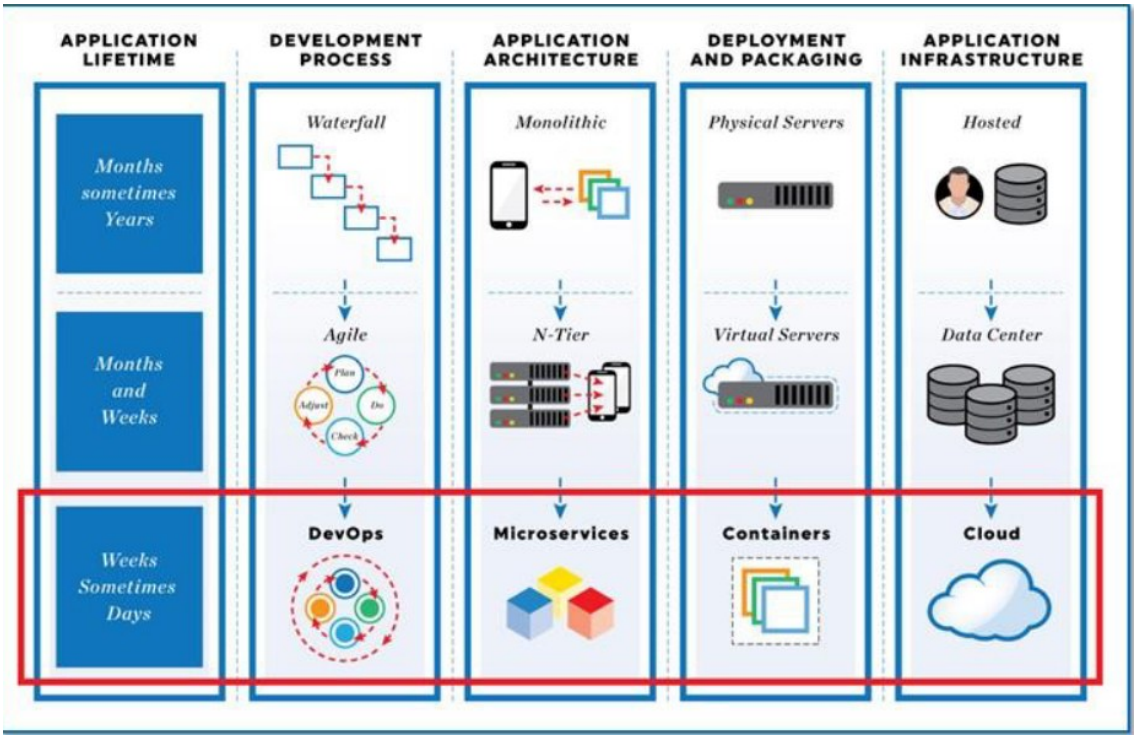

***Ilustración 12**. Transición de Waterfall a DevOps.*

En la ilustración 13 se encuentra un resumen de los 4 ejes DevOps con sus definiciones principales.

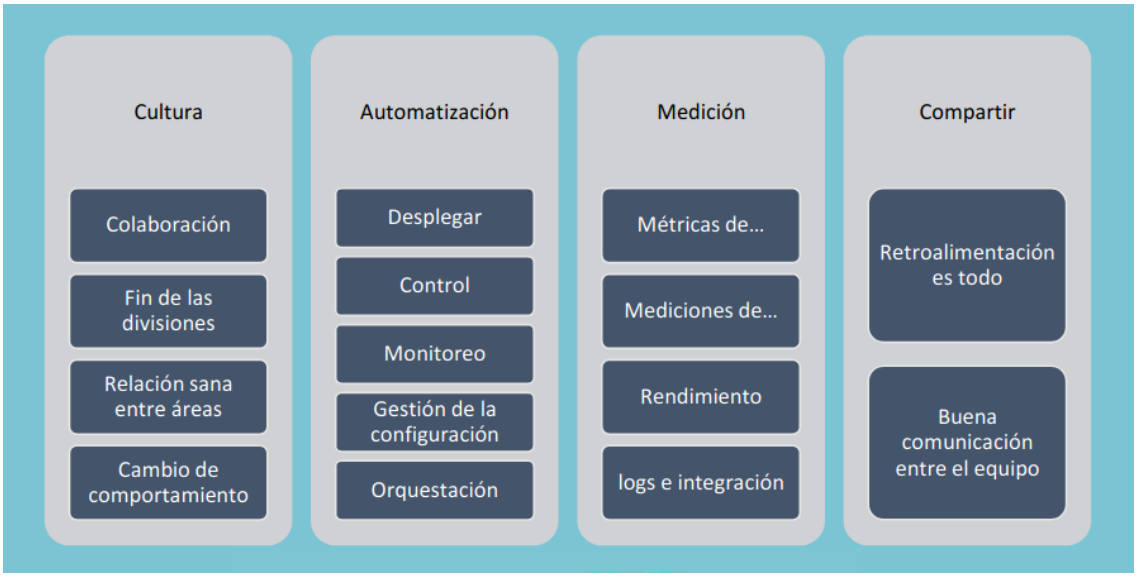

***Ilustración 13**. Ejes DevOps.*

Existen además características técnicas que la metodología DevOps debe soportar (Carvalho, Moreira & Rosso, s.f.):

1. Infraestructura como código
2. Orquestación de servidores
3. Gestión de la configuración
4. Aprovisionamiento dinámico de ambientes
5. Control de versiones entre infraestructura y desarrollo
6. Diferentes ambientes, por lo menos desarrollo, pruebas y producción.
7. Se debe hacer TDD.
8. El equipo de operaciones debe participar en etapas tempranas
9. Mediciones de eventos
10. Capacidad de respuesta ante incidentes y problemas
11. Copias de seguridad confiables

La seguridad debe ser un objetivo compartido, donde los desarrolladores reconocen la importancia de la seguridad y el equipo de seguridad reconoce la importancia de las entregas rápidas, y juntos hacen un proceso que satisface ambas necesidades (Koskinen, 2020). Por esta razón, toma cada vez más fuerza el concepto de DevSecOps que se define a continuación. En la ilustración 14 se observa una breve historia de DevOps que no contempla la seguridad como parte de su base.

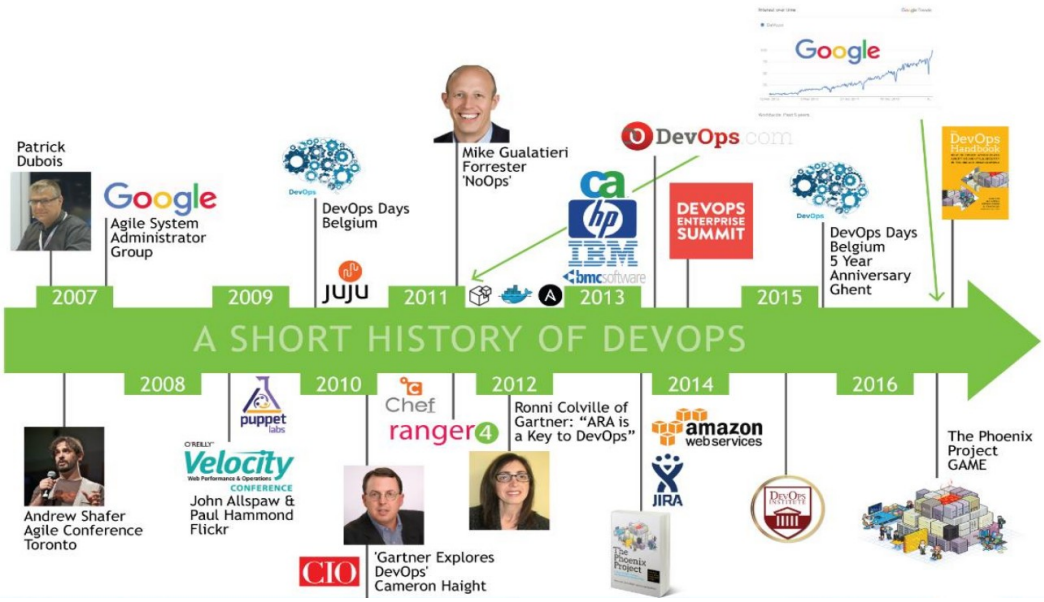

***Ilustración 14**. Historia de DevOps.*

2.1.5 DevSecOps

En compañías grandes, los departamentos de TI están organizados por función, esto hacía que en el pasado los departamentos de seguridad, operaciones y desarrollo fueran departamentos separados, desarrollo normalmente ha sido asociado a tareas de escribir aplicaciones de software, operaciones es responsable por mantener la infraestructura funcional y desplegar lo realizado por desarrollo normalmente en centros de datos, aunque en la última década, se puede hacer en la nube. Seguridad es el rol usado para asegurar la gobernanza y otros temas de cumplimiento.

Las técnicas utilizadas para proteger los sistemas tienen varios nombres según la bibliografía, pero pueden denominarse colectivamente, prácticas de seguridad de software, actividades de seguridad de software o métodos de diseño de seguridad de sistemas de información. La importancia de la seguridad del sistema crece constantemente a medida que más de nuestras vidas y dispositivos están conectados a la web (Koskinen, 2020).

Si bien la metodología de DevOps está bien desarrollada y se utiliza en la práctica dentro de la industria, la inclusión de medidas de seguridad en el ciclo de DevOps sigue siendo un desafío. Esta particular falta de enfoque en la seguridad en DevOps y otras metodologías de desarrollo de software ágiles y de ritmo rápido ha sido criticada durante un tiempo (Carter, 2017).

Los desafíos actuales a resolver en el proceso de inclusión de seguridad dentro de DevOps son los siguientes:

1. El enfoque actual de la seguridad se contempla fuera del ciclo de desarrollo de las aplicaciones, normalmente con pruebas que dependen de la experiencia del equipo de desarrollo, seguido por pruebas de penetración o hacking ético tratando detectar vulnerabilidades de manera interna o externa.
2. Muchos equipos de seguridad trabajan con una visión en la que sus objetivos se centran en inhibir el cambio tanto como sea posible.
3. Mientras los equipos de operaciones y desarrollo están en proceso de uso de tecnologías de vanguardia, los equipos de seguridad están centrados en batallas del ayer.
4. Los equipos de seguridad tradicional descubren las vulnerabilidades tarde en producción aumentando el costo para su resolución.
5. Es difícil el acercamiento del área de seguridad con desarrollo y operaciones.

El enigma de la seguridad es que se ve mejor cuando falta: es decir, un sistema no seguro puede generar titulares a través de ciberataques y filtración de datos, mientras que un sistema bien protegido pasa desapercibido. (Koskinen, 2020). Curiosamente, desde la perspectiva de los gerentes, la colaboración entre seguridad y los desarrolladores pueden parecer más fuertes de lo que realmente son. Según la investigación (Puppet, 2019) el 64% de la gerencia de nivel superior cree que el personal de seguridad está involucrado en el desarrollo de software, mientras que solo el 39% de los desarrolladores de software está de acuerdo con esta opinión. La incorporación de la seguridad en DevOps de una manera tan organizada se ha acuñado como DevSecOps, mientras que algunos prefieren SecDevOps o DevOps seguro. La ilustración 15 muestra los resultados de búsqueda de Google Trends para DevOps y DevSecOps (línea azul y rojo respectivamente).

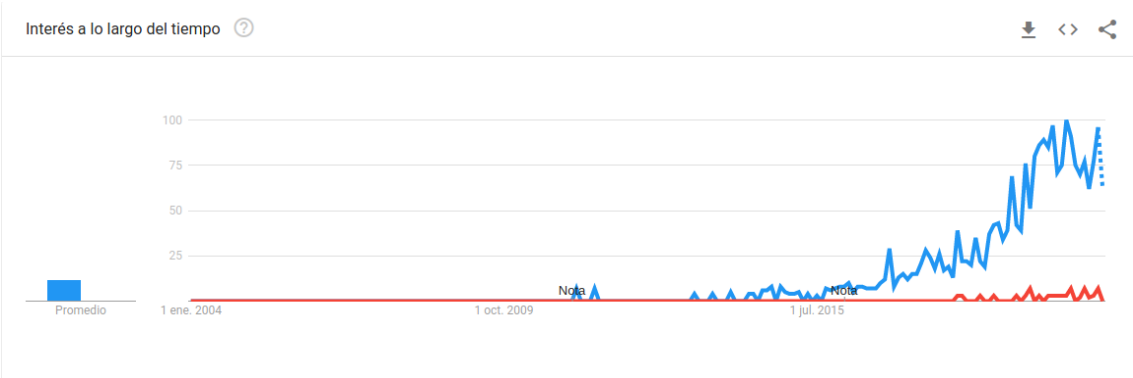

***Ilustración 15**. Adopción de DevSecOps Colombia.*

DevSecOps entonces se puede ver como una unión de desarrollo, seguridad y operaciones. Se puede considerar un proceso para ayudar a organizaciones a mejorar el tiempo de entrega y poder liberar software de forma rápida integrando el enfoque de seguridad en el ciclo de desarrollo.

A diferencia de DevOps en DevSecOps la ilustración 16 presenta un manifiesto creado desde el 2014 (Lietz, Shrikant et al, 2020), donde se tienen los siguientes puntos:

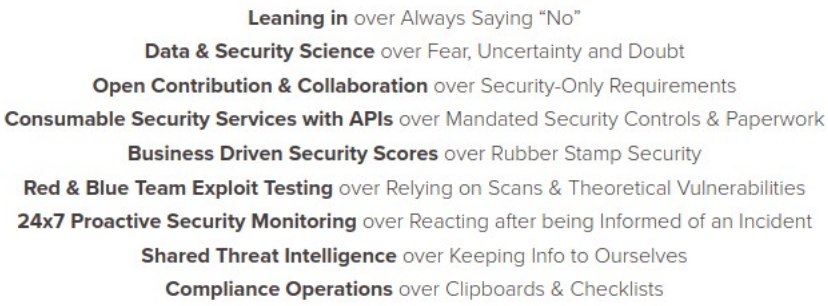

Ilustración 16*.* Manifiesto DevSecOps*.*

Se busca, en este concepto, introducir seguridad desde las primeras fases del ciclo de desarrollo intentando minimizar la cantidad de vulnerabilidades y el acercamiento de la seguridad a temas de núcleo del negocio.

El propósito final es enviar a producción automáticamente una aplicación o sistema apropiadamente seguro en cuestión de minutos. Teniendo claro, que la seguridad total no existe, se minimiza el riesgo (Carvalho, Moreira & Rosso, s.f.).

Igual que en DevOps la parte cultural y la gestión juegan un rol muy importante, nuestro objetivo es ser resilientes no tener el mito típico de la seguridad perfecta. Según Netflix la mejor forma de prevenir los fallos es fallar constantemente (Carvalho, Moreira, Rosso & Ibiri, 2020).

Para poder lograr el objetivo de minimizar el riesgo nos debemos basar en los mismos cuatro pilares de DevOps, de (Koskinen, 2020) obtenemos la siguiente tabla 1:

Tabla 1. *Construcción de seguridad basado en los pilares de DevOps*

Principio DevOps	Consejo práctico
Cultura	Desarrolle una cultura orientada a la seguridad
	Cree una cultura con un sentido de responsabilidades compartidas, donde Desarrollo, operaciones y seguridad trabajen juntos para un objetivo común.
	Priorice la reparación de defectos de seguridad
Automatización	Automatice la seguridad tanto como sea posible, pero asegúrese de que las herramientas elegidas sean

	aplicables al entorno y la tecnología.
	Garantice la seguridad de los pipelines
	Reconozca que la automatización mejora las actividades de seguridad manuales, no las reemplaza.
Medición	Desarrolle métricas de seguridad.
	Mida y monitoree la seguridad constantemente.
	Proporcione un circuito de retroalimentación a los desarrolladores y operaciones sobre cuestiones de seguridad.
Compartir	Compartir conocimientos sobre principios de seguridad.
	Desarrollar la gestión de incidentes e involucrar a los desarrolladores en la solución de problemas de seguridad.

Si bien no son los únicos parámetros a tener en cuenta, nos sirve como base para entender los principios fundamentales de la unión de DevOps + sec.
Gene Kim en sus libros The phoenix project y The DevOps Handbook describe los pilares fundamentales de DevOps conocido como “Three ways”, ahora debemos complementarlos con seguridad (ver ilustración 17).

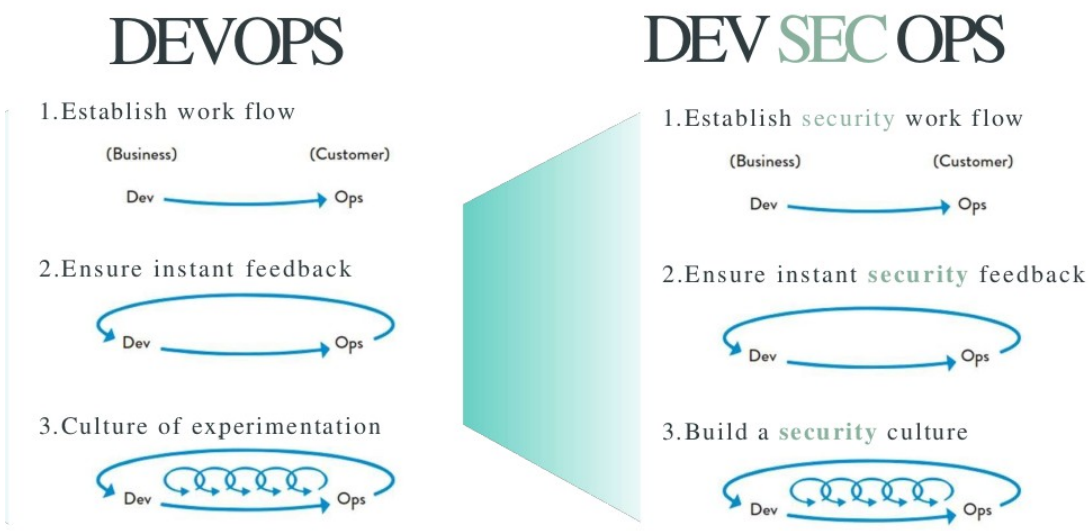

Ilustración 17. Three ways de DevOps a DevSecOps.

The first way Establecer flujo de trabajo de seguridad:

Tradicionalmente, la seguridad se ha realizado como una serie de tareas masivas imposibles de realizar que intentan abarcar todos los riesgos. Por ejemplo, escribir un conjunto completo de requisitos de seguridad, diseñar una arquitectura de seguridad integral, realizar una prueba de seguridad integral, etc.

The second way garantizar comentarios de seguridad instantáneos:

seguridad es una de las causas más comunes de la deuda técnica, y el costo de este trabajo aumenta dramáticamente a medida que avanza a través del ciclo de desarrollo de software. ¿Cómo mantenemos el trabajo de seguridad vigente? Para evitar este desafío, las organizaciones deben automatizar la seguridad tanto como sea posible y proporcionar información instantánea a los miembros del equipo que lo necesitan a través de las herramientas que ya están utilizando.

The Third Way construir una cultura de seguridad.

Muchas organizaciones tienen una cultura de seguridad de confianza ciega, culpa y ocultamiento que evita que los desarrolladores y las operaciones trabajen con seguridad. ¿Cómo creamos una cultura de seguridad? La clave de la cultura de DevSecOps es garantizar que la responsabilidad de la seguridad recae directamente en los equipos de desarrollo. En DevSecOps muchos de los procesos requeridos para el desarrollo de software, pruebas y despliegues son automatizados, por esto las verificaciones de seguridad deben ser implementadas por cada servidor y herramienta para reducir el número de posibles vectores de ataque.

Para poder definir políticas y aplicar DevSecOps debemos tener en cuenta algunos estándares de seguridad como lo es OpenScap, un estándar de seguridad mantenido por el NIST, este mantiene el proyecto OpenSCAP una colección de herramientas open source para implementar y reforzar este estándar. También existe el centro para la seguridad de internet (CIS) que nos provee guías en las configuraciones de los sistemas operativos, bases de datos, virtualización, frameworks y aplicaciones, también está el NCP donde podemos hacer una búsqueda de los documentos definidos por CIS (National Vulnerability Database, 2020).

Algunos ejemplos son CIS Docker Community Edition Benchmark (1.1.0), CIS Amazon Web Services Foundations Benchmark (v1.3.0), CIS VMware ESXi 5.5 Benchmark (1.2.0) que son consideradas buenas prácticas en la industria.

En la ilustración 18 encontramos el ciclo que deberíamos aplicar en DevSecOps.

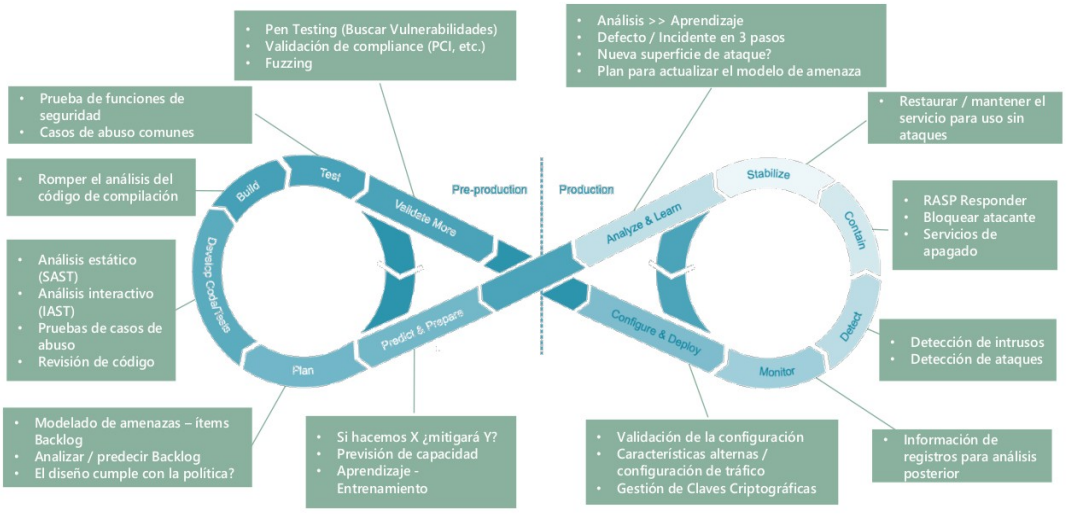

***Ilustración 18**.* Ciclo de desarrollo en DevSecOps*.*

La seguridad debe ser involucrada en un proceso más temprano donde la auditoria y cumplimiento serán parte fundamental, se debe invertir en el diseño, así como en entrenamiento y conciencia, para esto se debe trabajar hacia una visión y teniendo objetivos compartidos con la organización. Antes de describir cada uno de estos conceptos debemos abordar las infraestructuras ágiles.

2.1.6 Infraestructuras ágiles:

Las infraestructuras ágiles se pueden definir como guías de buenas prácticas creadas por las personas que trabajan en operaciones o administración de sistemas (sysadmin) basado en experiencias pasadas que implican la automatización y el uso de metodologías ágiles.

La automatización hoy en día está presente en entornos virtualizados, uso de nube, contenedores y aplicaciones serverless.
Para poder hacer automatización en infraestructuras ágiles hablaremos de dos conceptos, aprovisionamiento y gestión de la configuración, así como las herramientas a utilizar por cada parte del proceso.

2.1.6.1 Aprovisionamiento:

Es la manera de que el proceso de creación de nuevos entornos deba ser fácil y rápido buscando con ello minimizar el tiempo para que el consumidor lo pueda utilizar sin sacrificar la seguridad. Según estudios existen organizaciones donde la entrega de una máquina virtual puede tardar hasta 90 días para ser construida y utilizada en un entorno de pruebas (Carvalho, Moreira & Rosso, s.f.). El paso principal para ejecutar nuestras aplicaciones es elegir un sistema operativo donde ejecutar la aplicación de software construida, luego instalar las librerías o paquetes que acompañen a la misma y por último el cambio de parámetros por defecto basado en estándares mundiales de seguridad mencionados en el capítulo anterior, proceso que se debe llevar en minutos en lugar de semanas o incluso meses. Existen varias herramientas a utilizar según el caso de uso del equipo, estas se condensan en la tabla 2:

Tabla 2*.* Herramientas de Infraestructura como código

Herramienta	Creador	Comentarios	URL	Uso	Providers
Vagrant	Hashicorp	Aprovisiona máquinas desde un un archivo de de configuración declarativo que describe los requisitos de software, paquetes, configuración del sistema operativo.	https:// www.vagrantup.com/	Se puede utilizar para la creación de máquinas virtuales	VirtualBox, VmWare, docker, Hyper-v, Proveedor personalizado
Cloudformation	Amazon Web services	“Cloudformation le ofrece una forma sencilla de modelar un conjunto de	https:// aws.amazon.com/es/cloudformation/	Creación de recursos en la nube de AWS	AWScloudfr

		recursos relacionados de AWS y de terceros, aprovisionarlos de manera rápida y consistente y administrarlos a lo largo de sus ciclos de vida tratando la infraestructura como un código”			
Azure Resource Manager	Microsoft	“Permite implementar una aplicación repetidamente con la confianza de que los recursos se implementan de forma coherente. La infraestructura y las dependencias de la aplicación las define el usuario en una plantilla declarativa simple”	https://azure.microsoft.com/es-es/features/resource-manager/	Manejo de recursos de forma dinámica en Azure	Azure
Cloud Deployment manager	Google	“Especifica los recursos necesiarios para la aplicación en formato declarativo mediante yaml. También se usan plantillas Python o Jinja2 para parametrizar la configuración y permitir que su reutilizar paradigmas de despliegue común,”	https://cloud.google.com/deployment-manager?hl=es	Manejo de recursos de forma dinámica en GCP	Google Cloud Platform
Terraform	Hashicorp	Binario para crear infraestructura como código para aprovisionar y administrar cualquier nube mediante llamados de API.	https://www.terraform.io/	Definición de infraestructuras en múltiples proveedores de nube	Más de 100 proveedores y con la oportunidad de crear los propios.
Pulumí	Pulumí	Herramienta de infraestructura como código que permite escribir configuraciones de implementación utilizando su lenguaje de programación favorito.	https://www.pulumi.com/		Crear infraestructura como código en javascript, typescript, python,go, .net

La mayoría de las plantillas de IaC se crean mediante un proceso simple de tres pasos: diseño, codificación e implementación. Lo que pone en problemas a los equipos, pues la seguridad no es considerada en muchos casos.

De la misma forma que el código de la aplicación, las plantillas de IaC deben analizarse en busca de problemas de seguridad cada vez que se crean o actualizan (ver ilustración 19).

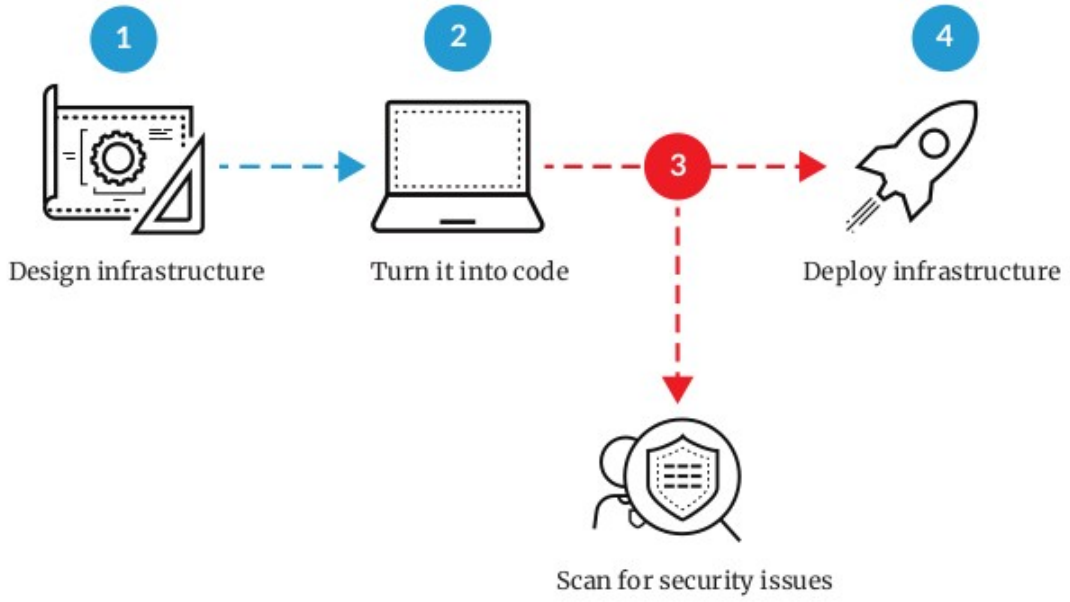

Ilustración 19*.* Ciclo ideal IaC*.*

El estudio (Palo alto networks, 2020) muestra que casi 200.000 plantillas de IaC son inseguras en su uso, el 43% de las bases de datos en la nube no están cifradas y el 60% de los servicios de almacenamiento en la nube tienen el log deshabilitado.

Gran parte se deben a errores comunes en la configuración de las plantillas que pueden ser prevenidos si se hace una verificación en etapas tempranas, proceso conocido como corrimiento a la izquierda.

Debido a la gran adopción de terraform mostrada en el estudio (Palo alto networks, 2020) el análisis de este proyecto se centrará en esta herramienta.

Con terraform se puede realizar la creación de ambientes dinámicos en diferentes proveedores, tanto de nube, como en infraestructuras on premises, con su forma de trabajo uno de los problemas básicos de seguridad es el manejo de secretos de los recursos a desplegar, esto pues, terraform crea un archivo de estado donde almacena la forma de la infraestructura después de cada ejecución, si algún delincuente informático obtiene este archivo tendría el acceso a información privilegiada. Otro error común según el estudio (Palo alto networks, 2020) es la creación de máquinas con grupos de seguridad abiertos o mala configuración de servicios como SSH. El tercer tema a resolver desde seguridad es la gestión de permisos e identidades para la cuenta que debe ejecutar terraform.

Si bien existen algunas organizaciones que han optado por utilizar servidores virtualizados en la nube, algunas otras desde la creación en docker en 2013 ven una

opción interesante y viable para ejecutar cargas de trabajo con esta nueva tecnología, pero de igual manera como en el paso anterior la seguridad no es tomada en cuenta o si se hace es de forma mínima, por esta razón el objetivo es analizar el enfoque de distroless, build packs apalancado por google y la comunidad, buscando de esta forma reducir la superficie de ataque.

Las diez imágenes de contenedores oficiales más populares tienen un número significativo de vulnerabilidades conocidas. Utilizar una imagen oficial no reemplaza la higiene de seguridad habitual (Miller & Zitzman, 2020). Para esto, se puede realizar la inclusión de un validador de imágenes a nivel de CI y así verificar las imágenes contra bases de datos con vulnerabilidades (ver ilustración 20).

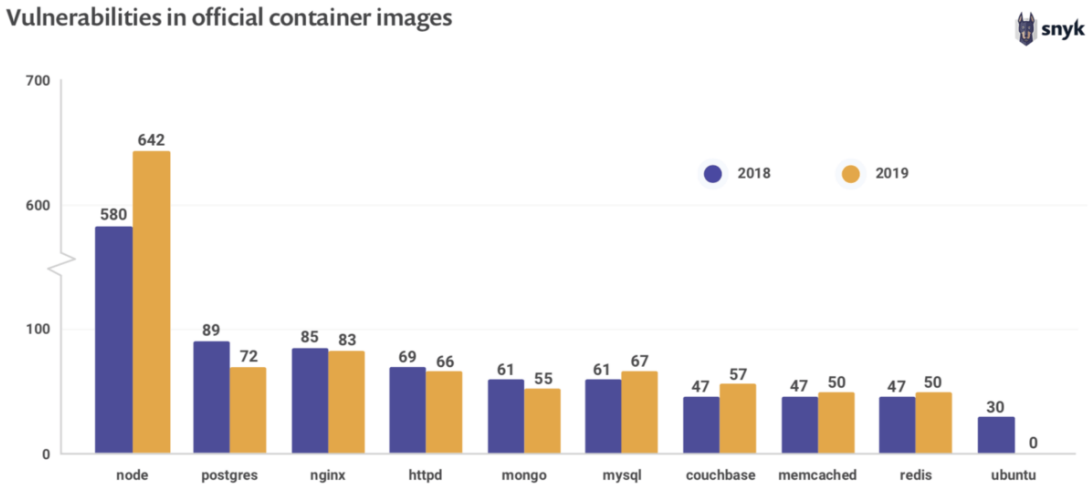

Ilustración 20. Vulnerabilidades en imágenes de contenedores.

Los nuevos enfoques de despliegues sugieren además el uso de infraestructuras serveless, la aplicación de seguridad en estos entornos se deja como tema de investigación para futuros trabajos, al igual que el enfoque de git ops.

2.1.6.2 Gestión de la configuración:

Esta se creó hace más de dos décadas por el profesor Mark Burgess quien se enfrentó con diferentes problemas para la creación de ambientes de laboratorio (Carvalho, Moreira & Rosso, s.f.).

El objetivo general de la gestión de la configuración como práctica es maximizar el grado en que los sistemas se ajustan a las expectativas predeterminadas, mientras se minimiza el costo de mantener el comportamiento de la red dentro de las pautas predeterminadas (Burgess & Couch, 2006).

Con la creación de CFEngine por parte de Burgess se podían definir estados para distintos sistemas operativos utilizando para ello la convergencia (llegar a un estado final deseado) por encima de la congruencia (construir un resultado a partir de un

estado en blanco), por 10 años entre 1993 y 2003 tanto el CFEngine como la gestión de la configuración evolucionaron a nuevas opciones que hoy en día se utilizan en la industria como es poderse curar automáticamente (self healing), hacer detección de anomalías y las comunicaciones seguras.

La idea de la gestión de la configuración es poder realizar un proceso de estandarización y obligar el cumplimiento de ciertas políticas en servicios utilizando para ello el principio de la automatización y el control de versiones.

Han surgido un par de herramientas más como lo son puppet, chef, salt stack y ansible que heredan varias características del CFEngine. En esta tesis se utilizará ansible para realizar la configuración y el endurecimiento de las máquinas.

Las prácticas de seguridad son diferentes para cada equipo y sistema, en mi perspectiva existen tres categorías: Endurecimiento del servidor y gestión de la configuración, endurecimiento de la aplicación, manejo de identidades.

Cabe aclarar que en este trabajo final de máster haré énfasis en los dos primeros pasos de forma técnica, la gestión y manejo de identidades no se abordará.

2.1.6.3 Endurecimiento del servidor:

Después de poner en funcionamiento el sistema operativo debemos aplicar prácticas para garantizar que nuestros sistemas minimizan el riesgo de ser vulnerados, para ello debemos modificar los comportamientos por defecto y correr algún software de verificación ampliamente utilizado en la industria. La idea de estas herramientas es comparar la máquina en ejecución frente a un conjunto de reglas ya definidas en el software y dar una salida que contenga las diferencias entre ambas.

Nuestra idea será definir una línea base que se considera un conjunto de atributos de un sistema en un punto del tiempo, el cual sirve como punto de referencia para cambios en el sistema. Nuestro enfoque se basará en distribuciones linux, específicamente en ubuntu 20.04 la última distribución LTS al momento de escritura de este trabajo.

En linux y contenedores existen diferentes herramientas que podemos utilizar para verificar si un servidor está cerca de la línea base definida o si debemos modificar ciertos parámetros para minimizar el riesgo (ver tabla 3).

Tabla 3. Comparación herramientas linux y docker

Nombre	Pruebas de cumplimiento	Aplica medidas	Automatización	URL
Bastille	NO	SI	NO	http://bastille-linux.sourceforge.net/
Tiger	NO	SI	NO	https:// www.nongnu.o

				rg/tiger/
Lynis	SI	NO	SI	https://cisofy.com/lynis/
OpenSCAP	SI	SI	SI	https://www.open-scap.org/
DockScan	NO	NO	SI	https://github.com/kost/dockscan
Docker bench security	SI	NO	NO	https://github.com/docker/docker-bench-security
Clair	NO	NO	SI	https://coreos.com/clair/docs/latest/
Anchore	NO	NO	SI	https://github.com/anchore/anchore-engine
Dagda	NO	NO	SI	https://github.com/eliasgranderubio/dagda

Podemos obtener un conjunto de buenas prácticas al aplicar seguridad en otros sistemas operativos diferentes al trabajo en cuestión (Miller, 2018).

2.1.7 Modelado de amenazas:

El modelado de amenazas es un proceso para capturar, organizar y analizar toda la información que afecta a la seguridad de una aplicación. Permite tomar decisiones informadas sobre los riesgos de seguridad en las aplicaciones. Además de producir un modelo, los esfuerzos para un modelado de amenazas típico también producen una lista priorizada de mejoras de seguridad en los requisitos, diseño e implementación de las aplicaciones.

El modelado de amenazas está emergiendo como el enfoque definitivo para el desplazamiento a la izquierda. Con un modelado de amenazas efectivo se puede encontrar, enumerar y corregir errores de seguridad en la etapa más temprana posible del ciclo de vida.

Existen tres modalidades clásicas:

1. Centrada en el atacante
2. Centrada en la arquitectura del sistema
3. Centrada en los activos.

El modelado de amenazas debe estar presente en todo el ciclo de vida del desarrollo de software, se busca que en las primeras etapas de planificación se tenga uno de alto nivel y se hagan refinamientos al memos con el avance del proyecto, esto pues se tendrán mayores detalles del sistema y por ende una exposición mayor a nuevos vectores de ataque.

El modelado de amenazas impacta de forma técnica nuestro negocio, esto debido a que la elección de una tecnología particular nos lleve a identificar la responsabilidad y las amenazas creadas por esa elección.

Hay varias formas de hacer modelado de amenazas que van desde metodologías formales (STRIDE, SEI OCTAVE, PASTA, TRIKE, PTA, CORAS), juegos de cartas (por ejemplo, OWASP Cornucopia) hasta sesiones informales de pizarra.

Estas son algunas herramientas libres u open source para hacer modelado de amenazas (ver tabla 4):

Tabla 4. Herramientas para modelado de amenazas

Nombre herramienta	URL
OWASP thread dragon	https://owasp.org/www-project-threat-dragon/
Microsoft Threat Modeling Tool	https://docs.microsoft.com/en-gb/azure/security/develop/threat-modeling-tool
threatspec	https://threatspec.org/
Threat playbook	https://we45.gitbook.io/threatplaybook/
Owasp cornucopia	https://owasp.org/www-project-cornucopia/

2.1.8 Endurecimiento de la aplicación:

En el informe (Schleen & Weeks, 2020). el 24% de los encuestados confirmaron o sospecharon de infracciones relacionadas con sus prácticas de desarrollo de aplicaciones, aunado a lo anterior, el estudio de Horvath, Zumerle. & Gardner (2020). brinda algunos supuestos de planificación estratégica como los son

1. Para el 2025, el 70% de los ataques contra contenedores provendrán de vulnerabilidades conocidas y configuraciones incorrectas que podrían haberse solucionado.
2. Para el 2025, las organizaciones acelerarán la corrección de las vulnerabilidades de codificación identificadas por SAST en un 30% con sugerencias de código aplicadas desde soluciones automatizadas, en comparación con menos del 1% actual reduciendo el tiempo dedicado a la corrección de errores en un 50%.
3. Para 2024, la provisión de una lista de la composición de software detallada y actualizada regularmente por parte de los proveedores de software será un requisito no negociable para al menos la mitad de los compradores de software empresarial, en comparación con menos del 5% en 2019.

Un riesgo adicional es que el código abierto constituye el 70% de las bases de los proyectos auditados (Synopsys, 2020), el problema no es en sí el uso de las mismas sino la poca verificación de la seguridad de lo utilizado (Ver ilustración 21).

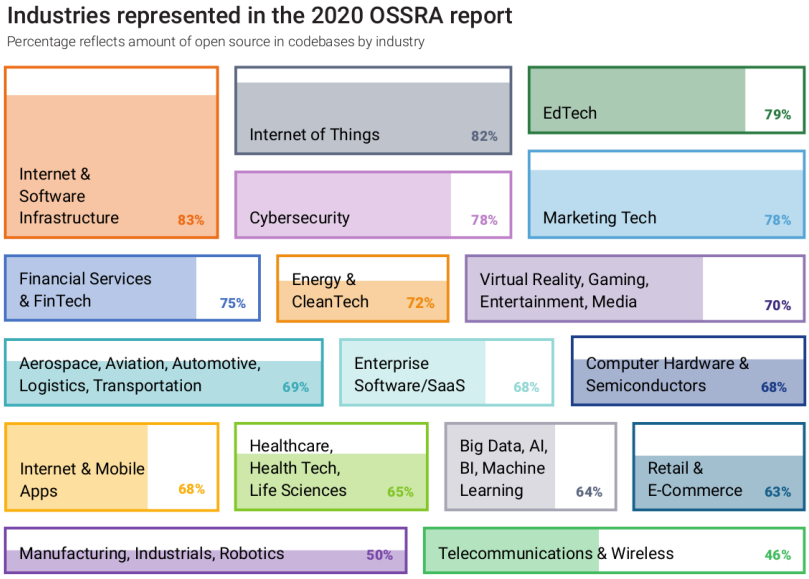

Ilustración 21. Cantidad de librerías open source en códigos base por industria.

Los errores y las debilidades en el software son comunes, el 84% de las brechas de software explotan vulnerabilidades en la capa de aplicación donde entre un 70 y 80 por ciento de los ataques exitosos suceden por vulnerabilidades conocidas (Carvalho, Moreira, Rosso & Ibiri, 2020).

2.1.9 Herramientas de prueba de seguridad en aplicaciones

Las herramientas de prueba de seguridad en aplicaciones se pueden representar en la ilustración 22:

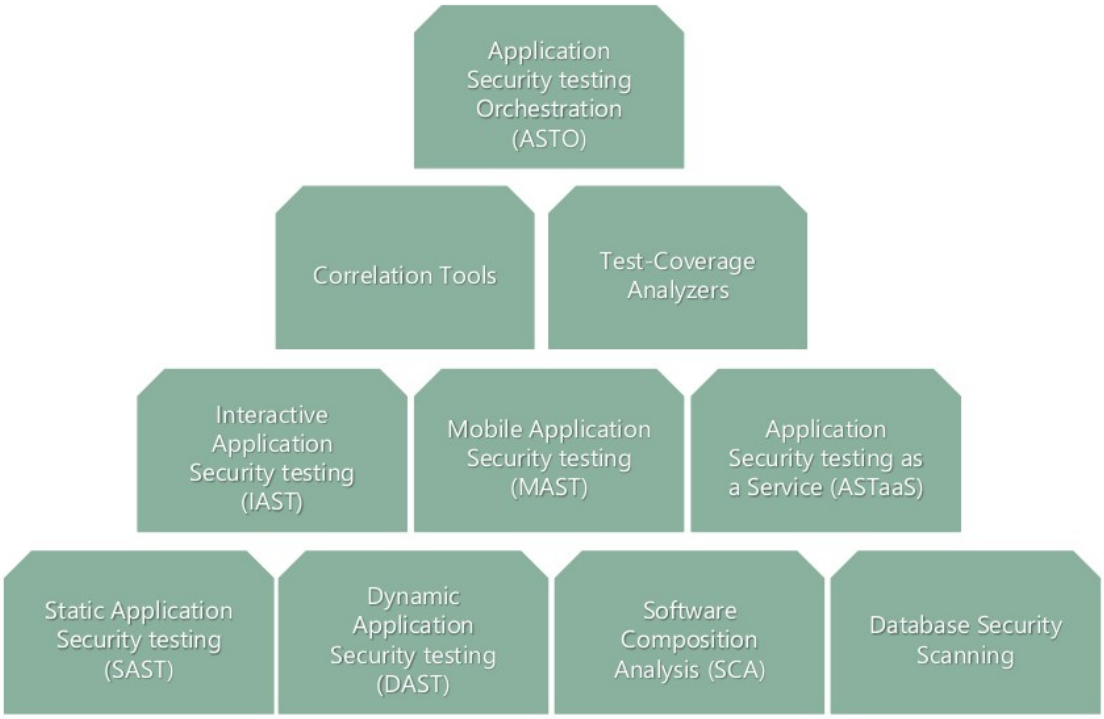

Ilustración 22. Pirámide AST.

2.1.9.1 Detección de secretos

Los equipos de desarrollo y operaciones de las grandes organizaciones utilizan miles de secretos como claves API, token, llaves de acceso, credenciales para interconectar los componentes desarrollados. El riesgo es que estos secretos se encuentran por todas partes lo que es conocido como “secret sprawl”.

El propósito es realizar la detección automática de secretos antes de que se encuentren en lugares no debidos como lo es una wiki, documentos no confidenciales o peor aún en el sistema de control de versiones.

Empresas como Uber han sufrido errores de este estilo, la primera de ella en mayo 12 del 2014 donde un atacante pudo acceder al almacén de datos de S3 de la empresa utilizando una clave de acceso que se hizo pública con permisos administrativos completos. La segunda fue en octubre 13 del 2016 por publicación de una clave de acceso de AWS en github (Federal trade commission, 2018).

En (Meli, McNiece & Reaves, 2019). se tiene información importante del manejo de secretos en github, fueron detectadas 201.642 credenciales en un periodo de tiempo de estudio de seis meses. Estos secretos sufrieron brecha accidental no intencional. El estudio estima que el 89.10% de todos los secretos es sensible, 93.74% de los

secretos de APIS y 76.24% de las llaves asimétricas. La investigación concluye además que le ocurre a todo el mundo, con cada tipo de proyecto independientemente de la experièncía (ver ilustración 23).

Secret	# Total	# Unique	% Single-Owner
Google API Key	212,892	85,311	95.10%
RSA Private Key	158,011	37,781	90.42%
Google OAuth ID	106,909	47,814	96.67%
General Private Key	30,286	12,576	88.99%
Amazon AWS Access Key ID	26,395	4,648	91.57%
Twitter Access Token	20,760	7,935	94.83%
EC Private Key	7,838	1,584	74.67%
Facebook Access Token	6,367	1,715	97.35%
PGP Private Key	2,091	684	82.58%
MailGun API Key	1,868	742	94.25%
MailChimp API Key	871	484	92.51%
Stripe Standard API Key	542	213	91.87%
Twilio API Key	320	50	90.00%
Square Access Token	121	61	96.67%
Square OAuth Secret	28	19	94.74%
Amazon MWS Auth Token	28	13	100.00%
Braintree Access Token	24	8	87.50%
Picatic API Key	5	4	100.00%
TOTAL	**575,456**	**201,642**	**93.58%**

Ilustración 23. Secretos en github.

También gitlab descubrió que el 18% de los proyectos alojados en esa plataforma contienen secretos siendo el tercer tipo de vulnerabilidad más común (Haber, 2020).

La tabla 5 muestra el listado de herramientas de análisis de secretos opensource o libres:

Tabla 5. Herramientas para detección de secretos

Nombre herramienta	URL
Talisman	https://github.com/thoughtworks/talisman
GitGuardian	https://www.gitguardian.com/
Git secret	https://git-secret.io/
Gitleaks	https://github.com/zricethezav/gitleaks
TruffleHog	https://github.com/dxa4481/truffleHog
Git hound	https://github.com/tillson/git-hound
gittyleaks	https://github.com/kootenpv/gittyleaks
git-secrets	https://github.com/awslabs/git-secrets
Repo Supervisor	https://github.com/auth0/repo-supervisor

En la tabla 6 se relaciona el listado de herramientas para la gestión de secretos opensource o libres:

Tabla 6. Herramientas para gestión de secretos

Nombre herramienta	URL
Hashicorp vault	https://www.vaultproject.io/
Keywhiz	https://square.github.io/keywhiz/
Mozilla sops	https://github.com/mozilla/sops
Confidant	https://lyft.github.io/confidant/
Barbican	https://wiki.openstack.org/wiki/Barbican

2.1.9.2 SAST:

La inspección de Fagan (1976) dió paso al análisis estático de código como alternativa al aumento de la seguridad del software, un proceso que se realiza sobre el código de una aplicación sin necesidad de ejecutarse, normalmente se hace de forma automatizada con algunas herramientas, pero se podría tener en cuenta el análisis estático manual realizado por un humano. La idea principal es encontrar defectos de codificación en el ciclo de vida de desarrollo, por esta razón los desarrolladores podrán configurar su IDE para estas herramientas.

En la actualidad estas herramientas de análisis de código estático enfocadas en mejorar la seguridad del software, se han optimizado y la mayoría utilizan patrones para encontrar vulnerabilidades, combinando técnicas basadas en modelos de verificación y patrones de texto, acoplando así métodos utilizados en los programas de detección de errores y de revisión de estilo (Ospina, 2015).

La tabla 7 muestra el listado de herramientas SAST opensource o libres:

Tabla 7. Herramientas para SAST

Nombre herramienta	URL	Lenguajes soportados
Bandit	https://github.com/PyCQA/bandit	Python
Agnitio	https://sourceforge.net/projects/agnitiotool/	ASP, ASP.NET, C\#, Java, Javascript, Perl, PHP, Python, Ruby, VB.NET, XML
.NET Security Guard	https://security-code-scan.github.io/	.NET, C\#, VB.net
Brakeman	https://brakemanscanner.org/	Ruby
Deep Dive	https://discotek.ca/deepdive.xhtml	Java

RIPS	http://rips-scanner.sourceforge.net/	PHP
PHP code sniffer	https://github.com/squizlabs/PHP_CodeSniffer	PHP
Sonarqube	https://www.sonarqube.org	*Java, JavaScript, C#, TypeScript, Kotlin, Ruby, Go, Scala, Flex, Python, PHP, HTML, CSS, XML and VB.NET*
nodejsscan	https://github.com/ajinabraham/nodejsscan	Nodejs
Flaw finder	https://github.com/david-a-wheeler/flawfinder	C/C++
Go Sec	https://github.com/securego/gosec	Go
Sobelow	https://github.com/nccgroup/sobelow	Elixir
SpotBugs	https://spotbugs.github.io/	Groovy (Ant, Gradle, Maven y SBT)
Puma scan	https://github.com/pumasecurity/puma-scan	.NET
npm-check	https://github.com/dylang/npm-check	nodejs
graudit	https://github.com/wireghoul/graudit	Java, PHP, Python
Eslint	https://eslint.org/	Javascript
Typescript eslint	https://github.com/typescript-eslint/typescript-eslint	Typescript

2.1.9.3 Análisis de composición de software SCA:

Las herramientas SCA examinan el software con el objetivo de determinar los orígenes de todos los componentes y librerías utilizados dentro del software. Este tipo de herramientas son muy útiles para encontrar vulnerabilidades en componentes populares, especialmente en código abierto comparando frente a bases de datos de vulnerabilidades así como en restricciones de licencias en nuestro código.

El deseo universal de una innovación más rápida exige una reutilización eficiente del código, lo que a su vez ha llevado a una creciente dependencia en las bibliotecas de software para código abierto y terceros. Estos artefactos sirven como bloques de construcción reutilizables, que se introducen en repositorios públicos (npm, Maven Central, PyPI, NuGet Gallery, RubyGems, etc.) donde se encuentran libremente, con el soporte de millones de desarrolladores en el globo terráqueo (ver ilustración 24).

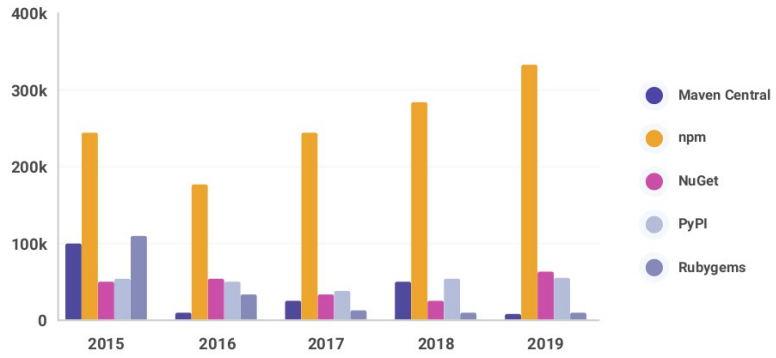

Ilustración 24. Creación de paquetes nuevos por repositorio público.

La identificación de licencias, los procesos para parchear vulnerabilidades conocidas y las políticas para abordar los paquetes de código abierto desactualizados y no compatibles son todos necesarios para el uso responsable del código abierto.

En el estudio realizado en Synopsys (2020) se examinaron 1253 aplicaciones en las cuáles se encontró que el código abierto constituye el 70% de las bases de código auditadas, lo que lo convierte en punto fundamental para revisar.

Ilustración 25. SCA.

En la ilustración 25 podemos ver que muchos códigos base tienen vulnerabilidades, y el 49% son de alto riesgo, a esto debemos sumar que algunas librerías son antiguas o ya no son mantenidas por la comunidad.

En el owasp top 10 de 2017 se define el uso de componentes con vulnerabilidades conocidas como la novena más aprovechada por los atacantes, pues, es sencillo obtener exploits para aquellas ya conocidas. Por esta razón, debemos ser cuidadosos al realizar la elección de alguna librería a utilizar en un programa, así como definir una

política empresarial donde se contenga el proceso para agregar una librería al código (ver ilustración 26 y 27).

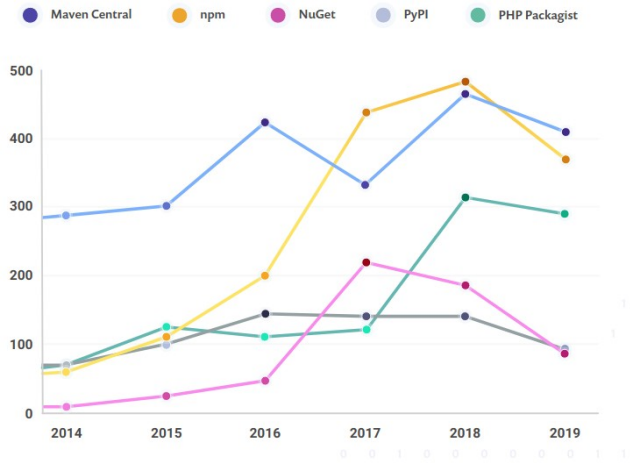

Ilustración 26. Vulnerabilidades por repositorio público.

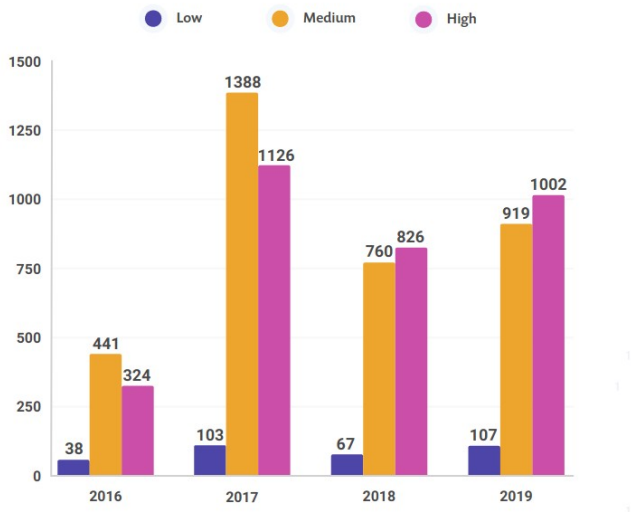

Ilustración 27. Vulnerabilidades por gravedad.

En (SAFECode security engineering training, 2020; Zimmermann, Staicu, Tenny, & Pradel, 2019) se hace un estudio de las amenazas en npm que puede ser la base para estudios posteriores con otros ecosistemas de software.

En la tabla 8 se muestra el listado de herramientas SCA opensource o libres:

Tabla 8. Herramientas para SCA

Nombre herramienta	URL	Lenguajes soportados
Owasp dependency check	https://owasp.org/www-	Java, .NET soporte

	project-dependency-check/	completo, analizadores experimentales para IOS, Python, PHP, NodeJs y ruby
Requires.io	https://requires.io/	Python
Retire.js	https://retirejs.github.io/retire.js/	Javascript
piprot	https://github.com/sesh/piprot	Python
Snyk	https://github.com/snyk/snyk	Javascript, .NET, Java, Python, Ruby, Go, php
Clearly defined	https://clearlydefined.io	Java
Oss review tolkit	https://github.com/oss-review-toolkit/ort	JS, Ruby, Rust, IOS, C, C++, Go, .Net, Java, Nodejs, PHP, Python, Dart/Flutter, Scala, Haskell
Safety	https://pypi.org/project/safety/	Python
dotnet-retire	https://github.com/RetireNet/dotnet-retire	.Net
Sonatype nexus	https://github.com/sonatype/nexus-public	Go, Ruby, Python, R, Java, JS, Docker, Helm

2.1.9.4 DAST:

Las herramientas DAST se ejecutan en código operativo para detectar problemas con interfaces, solicitudes, respuestas, scripts, inyección de datos, sesiones, autenticación y más. Las herramientas DAST emplean técnicas de Fuzzing: lanzar casos de prueba conocidos no válidos e inesperados en una aplicación, a veces en gran volumen (Carvalho, Moreira, Rosso & Ibiri, 2020).

Su propósito no es la eliminación de las pruebas manuales, es combinarlo junto a DAST para minimizar la superficie de impacto, existen enfoques que nos indican la combinación de DAST con pruebas End to END, combinando zap con robot framework o cypress por ejemplo.

Al momento de establecer la automatización debemos conocer la pirámide de pruebas definidas en Cohn (2009), que nos dará las pautas para aplicar procesos manuales o automáticos.

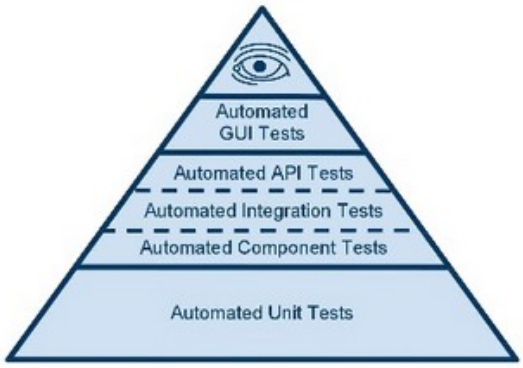

Ilustración 28. Pirámide de Cohn.

Nota: La pirámide se debe abordar de abajo hacia arriba, es decir empezando con los test unitarios y finalizando con pocas pruebas manuales.

Las pruebas de seguridad automatizadas en forma de SAST, SCA, DAST, análisis de secretos se están convirtiendo en la norma incluso para las empresas más grandes. Aprender a implementarlas de manera efectiva y elegir las herramientas correctas dependerá del caso de uso, convirtiéndose en un factor diferencial entre prácticas maduras e inmaduras de la inclusión de seguridad en DevOps (ver ilustración 29).

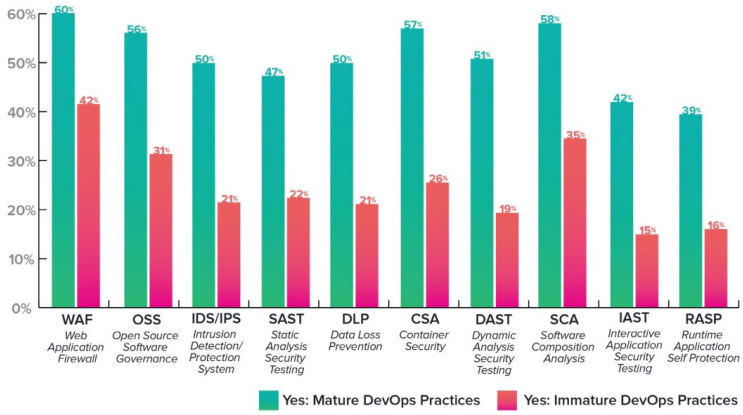

Ilustración 29. Integración de seguridad en DevOps.

En la tabla 9 se presenta el listado de herramientas DAST opensource o libres:

Tabla 9. Herramientas para DAST

Nombre herramienta	URL
Owasp ZAP	https://owasp.org/www-project-zap/
Arachni	https://www.arachni-scanner.com/
W3af	http://w3af.org/
Wapiti	https://wapiti.sourceforge.io/

En el estudio Mburano & Weisheng (2018) se encuentra una comparación para poder tomar una decisión de qué herramienta utilizar.

2.1.9.5 Gestión de vulnerabilidades y correlación:

Uno de los más grandes desafíos cuando incluimos seguridad en pipelines es el proceso de centralización de la información de los resultados obtenidos en cada una de las etapas. Son conocidas también como herramientas ASTO Application Security Testing Orchestration y se plantean como tema de estudio en trabajos de grado futuros.

Además de la recolección de la información, la verificación de falsos positivos es un gran problema en la seguridad de las aplicaciones, es con este objetivo que las herramientas de correlación aparecen (ver tabla 10), con el fin de disminuir el ruido ayudando con la validación y priorización de los hallazgos.

Tabla 10. Herramientas para gestión de vulnerabilidades y correlación

Nombre herramienta	URL
Archery	https://www.archerysec.com/
Jack hammer	https://github.com/olacabs/jackhammer
Defect Dojo	http://w3af.org/
Faraday	https://faradaysec.com/
WatchDog	https://github.com/flipkart-incubator/watchdog
OpenVAS	https://sectools.org/tool/openvas/
Faraday	https://github.com/infobyte/faraday

Existen algunas herramientas adicionales que si bien no son open source pueden servir como base para estudios futuros https://www.orchestron.io/, https://threadfix.it/, https://kondukto.io/, https://www.zeronorth.io/, https://qualitestgroup.com/, https://codedx.com/, https://about.gitlab.com/solutions/dev-sec-ops/

2.1.10 Cultura y gestión en DevSecOps

Existe un libro llamado culture eats strategy for lunch, su traducción sería como “La cultura se come la estrategia en una comida”, esta es una frase que hoy las personas participando en DevSecOps sienten como real, esto significa que, si no existe una alineación de sus miembros, el poner a funcionar herramientas no generará un valor a largo plazo y seguiremos viendo a seguridad como un palo en la rueda.

En este cambio cultural se busca que los equipos que integren DevSecOps sean responsables del cumplimiento, seguridad, costo y ante todo autónomos, pensando en un principio que todos los integrantes son responsables de la seguridad, para esto debemos pensar en las siguientes características:

1. Alta cooperación
2. Responsabilidades compartidas al igual que los riesgos
3. Entrenamiento continuo
4. Una falla es un descubrimiento para mejorar
5. Evitar el señalamiento
6. La innovación es continua
7. El compromiso es de todos

En Haber (2020) se puede encontrar 20 capacidades importantes necesarias para aprovechar al máximo DevSecOps. Esta puede ser una guía base para calificar el equipo y saber el estado de madurez de la metodología, también brinda un conjunto de recomendaciones para mejorar la puntuación.

El estudio se centra en 6 características, velocidad, prueba de eficiencia del proceso y desplazamiento a la izquierda, métodos de colaboración, nivel de automatización, cultura en seguridad, prácticas estandarizadas.

3. Desarrollo de la solución técnica

Para cumplir con los objetivos del trabajo final de máster, se define una solución técnica que incluirá la creación de los recursos tecnologicos como código, específicamente la creación tanto de máquina en local con vagrant, así como la construcción de una máquina en el proveedor de nube AWS con terraform, se hará un endurecimiento del servidor repetible con ansible que será llamado desde una rutina de packer. En el apartado de almacenamiento y verificación de código en el pipeline se utilizará gitlab CI en su versión community, se utilizarán varias herramientas open source en la verificación de cada uno de los pasos, una para análisis de secretos, otra para análisis de dependencias, una más para hacer SAST, todo esto basado en los principios de integración continua y despliegue continuo. Como resultado se podrán obtener artefactos de seguridad que serán el insumo para las personas involucradas en el proyecto, lo que dará una estrategia de hacking continúo, esto debe estar alineado a algun software de notificaciones empresarial o a un tablero de control para poder utilizar métricas como el tiempo medio para descubrir vulnerabilidades comparado frente al tiempo medio para la resolución de las mismas, lo interesante del enfoque más allá de la herramienta a ejecutar es la capacidad de alinear estos resultados con un proceso que permita gestionar los riesgos introducidos en el código o la infraestructura.

3.1 Tecnologías empleadas:

- **Servidor a utilizar:**

Ilustración 30. Logo Ubuntu.

La aplicación seleccionada necesita un servidor para su ejecución, como parte fundamental del proceso, con el advenimiento de la nube, la creación y destrucción de instancias. es un proceso natural que debe alinearse a una estrategia definida de infraestructura como código. Al momento de escritura de este trabajo final de máster la última versión LTS de ubuntu (ver ilustración 30) es la 20.04, según el ciclo de actualizaciones de canonical existirá una versión LTS cada dos años en abril, la ventaja de este tipo de distribuciones es el soporte y las actualizaciones de seguridad extendidas. Adicional a las versiones LTS canonical publica versiones provisionales del mismo, cada seis meses, tres entre versiones LTS, de esta forma las versiones obtienen un nombre clave para su desarrollo como lo son Xenial Xerus, Bionic beaver y focal fossa, siendo estas las tres últimas versiones LTS, esto es importante conocerlo

pues establecerá las bases para los paquetes a utilizar (Canonical, 2020). Si se desea saber la versión de una versión de ubuntu especifica se puede utilizar el comando lsb_release -a.(Ver ilustración 31).

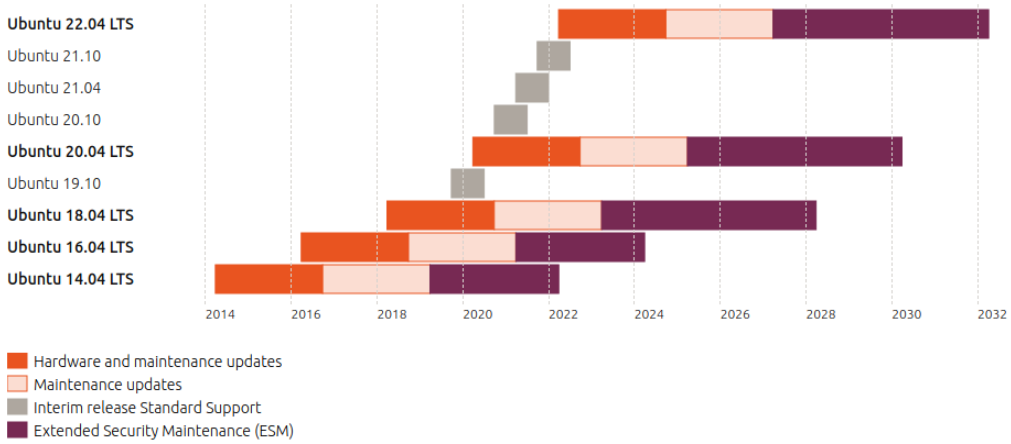

Ilustración 31. Ciclo de actualizaciones Ubuntu.

Para cerrar un poco la superficie de ataque siempre que utilicemos versiones de ubuntu debemos intentar elegir una versión de Ubuntu LTS y ojalá la última disponible ya que contiene correcciones de seguridad a vulnerabilidades detectadas, cabe aclarar que antes de realizar una migración de este estilo se deben hacer pruebas a los servicios ejecutando allí para evitar problemas de disponibilidad y compatibilidad.

- **Repositorio de código y CI:**

Ilustración 32. Logo Gitlab.

Gitlab (ver ilustración 32) cuenta con dos versiones, la versión community y la versión empresarial, en la versión community tendremos que regirnos por el tipo de licencia MIT EXPAT (está será la versión a utilizar en este TFM) la diferencia con la versión empresarial radica en que se tienen características adicionales y licencia propietaria.

GitLab se define como una plataforma DevOps completa, que puede ser accedida como una sola aplicación, minimizando de esta forma la complejidad de integración de diferentes herramientas, lo que se ve reflejado en la velocidad de los equipos y en la administración de los mismos.

Ilustración 33. Servicios Gitlab.

Si bien gitlab es ampliamente conocida como una plataforma para la gestión y almacenamiento de código, desde el año 2016 se han incluido nuevos servicios (ver ilustración 33) para incluir la parte de seguridad como un pilar fundamental de la plataforma.

- **Creación de entorno local:**

Ilustración 34. Logo Vagrant.

Vagrant (ver ilustración 34) es una herramienta utilizada para crear entornos de desarrollo repetibles utilizando para ello un archivo de configuración declarativo en donde se define la estructura de las máquinas, en este trabajo final de máster se utilizará virtualbox como virtualizador, si por alguna razón el lector no desea utilizar vagrant y quiere realizar la configuración directamente en el proveedor de nube puede omitir esta sección.

- **Creación de entorno de nube:**

Ilustración 35. Packer - Ansible.

Packer (ver ilustración 35) permite automatizar la creación de cualquier tipo de imagen a utilizar, la idea es crear una imagen con los requerimientos necesarios cumpliendo con la gestión de la configuración moderna, aunado a lo anterior, la configuración de los servidores debe ser repetible, y es acá, cuando utilizar ansible genera un gran valor, con ansible se puede realizar el aprovisionamiento de los servidores que incluye, instalación de paquetes, actualización de software y endurecimiento.

Ilustración 36. Terraform.

La idea de Terraform (ver ilustración 36) es poder crear infraestructuras heterogéneas utilizando para ello el concepto de infraestructura como código y de esta forma aprovisionar y administrar cualquier nube o servicio.

- **Verificación de seguridad en terraform**

Checkov Es una herramienta de análisis de código estático para IaC, para ello usa las mejores prácticas conocidas implementadas en manifiestos de IaC en Terraform, Cloudformation, kubernetes y serverless, liberada publicamente en diciembre del 2019, actualmente cuenta con +50 personas que contribuyen, más de 900.000 descargas y más de 1500 estrellas en github (Bridgecrew, 2020).

3.2 Instalación y uso de las herramientas:

3.2.1 Vagrant:

Para realizar la instalación de vagrant se descarga el binario de la página según el sistema operativo a instalar (Vagrant, s.f.) (ver ilustración 37).

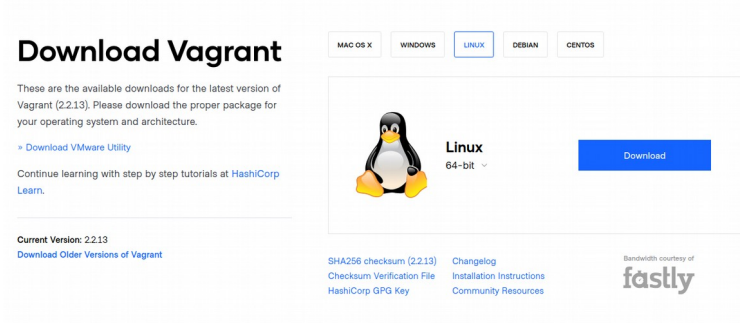

Ilustración 37. Descarga vagrant.

Se descarga un archivo comprimido en el cual se encuentra un binario, el paso siguiente es descomprimir el mismo y ponerlo en un path de ejecución para tenerlo disponible en todo el sistema operativo.

```
➜ unzip vagrant_2.2.13_linux_amd64.zip
Archive:  vagrant_2.2.13_linux_amd64.zip
  inflating: vagrant

~/Documents/TFM_Practice
➜ ls -ltr
total 26664
-rwxr-xr-x 1 ecastro ecastro 13787176 nov  6 22:24 vagrant
-rw-rw-r-- 1 ecastro ecastro 13509905 nov 14 20:42 vagrant_2.2.13_linux_amd64.zip
```

Ilustración 38. Descompresión vagrant.

Para validar que el binario está ejecutando verificamos la versión del mismo, que debe corresponder con la versión en el nombre de la ilustración 38 y 39:

```
➜ sudo cp vagrant /usr/bin
[sudo] password for ecastro:

~/Documents/TFM_Practice took 3s
➜ vagrant --version
Vagrant 2.2.13
```

Ilustración 39. Copia de vagrant a los binarios.

Para ejecutar una máquina con virtualbox como proveedor debemos crear un archivo Vagrantfile, este archivo tiene estructura específica que se aborda en la documentación (Vagrant, s.f.), en nuestro caso la idea es hacerlo lo más simple posible, definir la imagen a utilizar y el nombre de la máquina (ver ilustración 40).

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# Toda la configuración de vagrant se encuentra abajo. El número "2" en  Vagrant.configure
# le dice a vagrant que soporte esta versión. Vagrant tiene soporte para estilos antiguos o backwards compatibility.

Vagrant.configure("2") do |config|
    config.vm.box = "ubuntu/focal64"
    config.vm.define "host_1" do |host_1|
        host_1.vm.hostname = "edison"
    end
end
```

Ilustración 40. Archivo de vagrant

En la configuración se puede ver que la máquina a utilizar es focal64 que corresponde al nombre dado por ubuntu en la versión 20.04; 64 es la arquitectura sobre la que se ejecutará, en este caso 64 bits. Para validar la configuración utilizamos el comando vagrant validate Vagrantfile (ver ilustración 41)

```
~/Documents/TFM_Practice took 2s
→ vagrant validate Vagrantfile
Vagrantfile validated successfully.
```

Ilustración 41. Validación vagrant

Para poner en ejecución la máquina, debemos ejecutar el comando vagrant up, este comando pondrá nuestra máquina disponible, para validarlo podemos ir a virtualbox y confirmarlo (Ver ilustración 42 y 43).

```
→ vagrant up
Bringing machine 'host_1' up with 'virtualbox' provider...
==> host_1: Importing base box 'ubuntu/focal64'...
==> host_1: Matching MAC address for NAT networking...
==> host_1: Checking if box 'ubuntu/focal64' version '20201112.1.0' is up to date...
==> host_1: Setting the name of the VM: TFM_Practice_host_1_1605454237525_8285
==> host_1: Clearing any previously set network interfaces...
==> host_1: Preparing network interfaces based on configuration...
    host_1: Adapter 1: nat
==> host_1: Forwarding ports...
    host_1: 22 (guest) => 2222 (host) (adapter 1)
==> host_1: Running 'pre-boot' VM customizations...
==> host_1: Booting VM...
==> host_1: Waiting for machine to boot. This may take a few minutes...
    host_1: SSH address: 127.0.0.1:2222
    host_1: SSH username: vagrant
    host_1: SSH auth method: private key
    host_1: Warning: Connection reset. Retrying...
    host_1: Warning: Remote connection disconnect. Retrying...
    host_1:
    host_1: Vagrant insecure key detected. Vagrant will automatically replace
    host_1: this with a newly generated keypair for better security.
    host_1:
    host_1: Inserting generated public key within guest...
    host_1: Removing insecure key from the guest if it's present...
    host_1: Key inserted! Disconnecting and reconnecting using new SSH key...
==> host_1: Machine booted and ready!
==> host_1: Checking for guest additions in VM...
==> host_1: Setting hostname...
==> host_1: Mounting shared folders...
    host_1: /vagrant => /home/ecastro/Documents/TFM_Practice
```

Ilustración 42. Subida de la máquina con vagrant

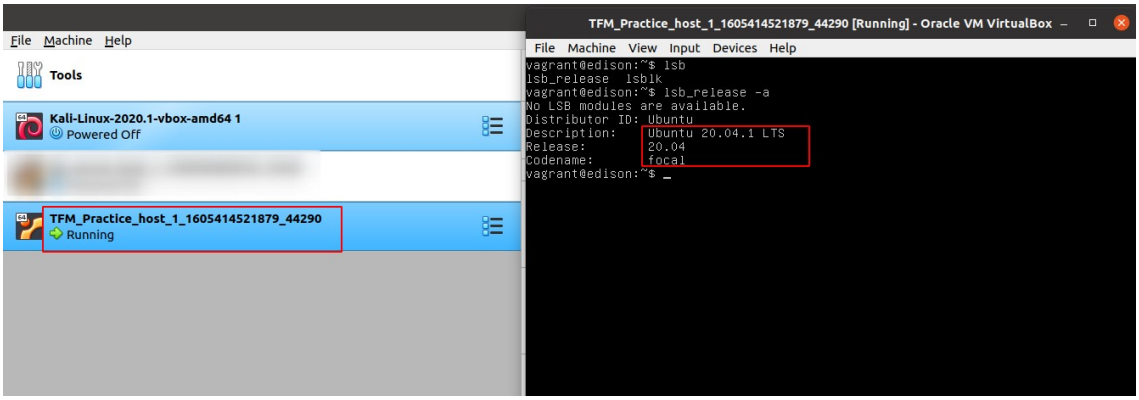

Ilustración 43. Máquina en ejecución con vagrant

Se creará por defecto un usuario llamado vagrant, para hacer login a la máquina podemos hacer vagrant ssh; Con la máquina recién instalada comenzaremos el proceso de configuración del servidor con ansible, esto buscando que el proceso sea repetible.

3.2.2 Ansible:

La instalación de ansible (ver ilustración 44) se puede realizar directamente desde los repositorios de ubuntu, para esto primero debemos actualizar la caché de los repositorios y los paquetes, para esto ejecutaremos el comando sudo apt update && sudo apt upgrade.

Para la instalación de ansible debemos ejecutar el comando sudo apt install ansible.

Para validar la instalación lo podemos hacer con ansible –version

```
➜ ansible --version
ansible 2.9.9
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/ecastro/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /home/linuxbrew/.linuxbrew/opt/python/lib/python3.7/site-packages/ansible
  executable location = /home/linuxbrew/.linuxbrew/bin/ansible
  python version = 3.7.7 (default, Apr 20 2020, 05:55:00) [GCC 5.4.0 20160609]
```

Ilustración 44. Verificación ansible.

Adicional a la instalación se creará una llave ssh que se pondrá en el servidor para realizar toda la configuración, en nuestro caso una llave de tipo ed25519, uno de los algoritmos más recomendados a hoy en día, este tipo de algoritmo es soportado desde la versión 6.5 de OpenSSH (ver ilustración 45).

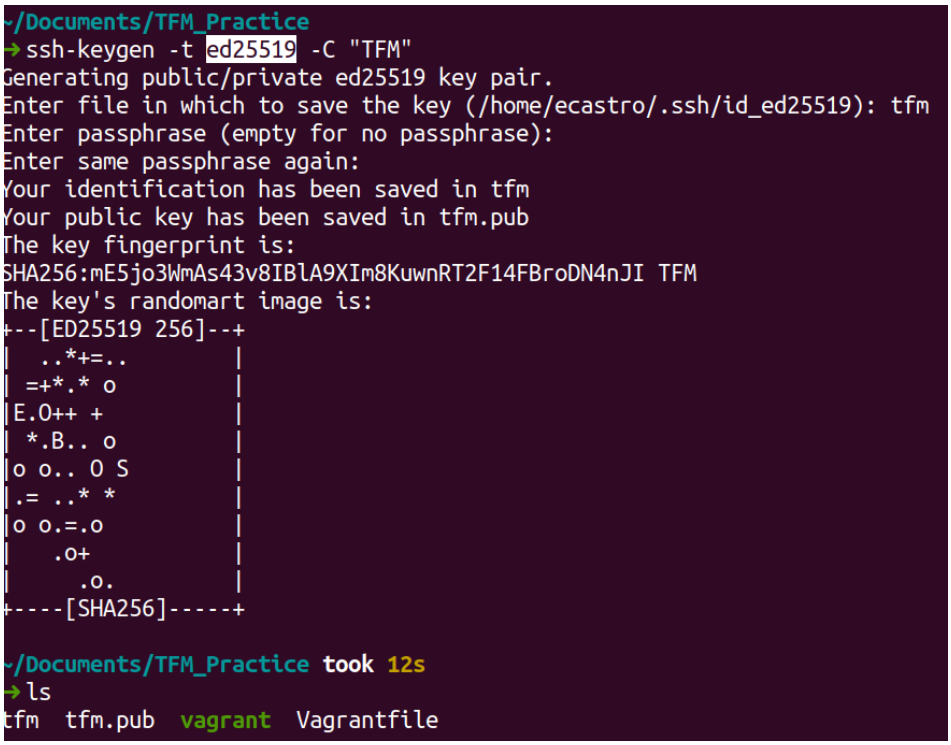

Ilustración 45. Generación de llave ed25519.

Para mejorar la creación de la llave, se podría utilizar el modificador -a para cambiar el número de rondas de KDF (Función de derivación de clave).

En el servidor a configurar haremos el proceso de creación del usuario ansible, para esto utilizamos el comando sudo adduser ansible (Ilustración 46).

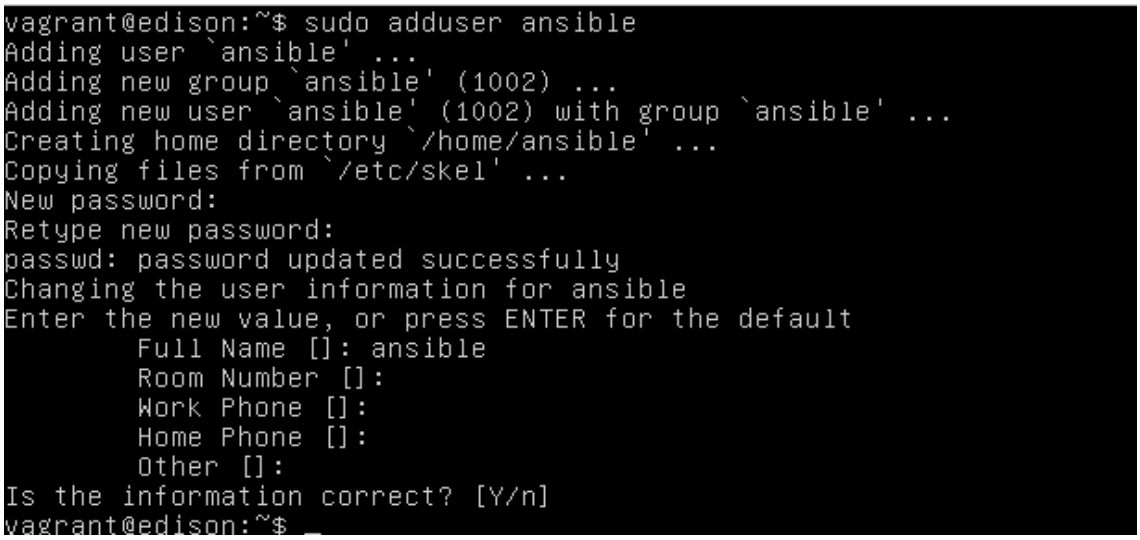

Ilustración 46. Creación de usuario ansible.

Para terminar la configuración del servidor, se configura un acceso sudo sin necesidad de contraseña (Ilustración 47):

```
vagrant@edison:~$ echo "ansible ALL=(ALL) NOPASSWD:ALL" | sudo tee /etc/sudoers.d/ansible
ansible ALL=(ALL) NOPASSWD:ALL
```

Ilustración 47. Acceso sudo ansible.

Se copia la llave de acceso al servidor con el comando ssh-copy-id -i tfm ansible@ip, para validar la conexión con el servidor se estable una sesión por ssh de la forma ssh -i tfm ansible@ip (Ilustración 48)

```
➜ ssh -i tfm ansible@192.168.0.18
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-53-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Sun Nov 15 16:28:52 UTC 2020

  System load:  0.0               Processes:               115
  Usage of /:   3.4% of 38.71GB   Users logged in:         1
  Memory usage: 18%               IPv4 address for enp0s3: 192.168.0.18
  Swap usage:   0%


1 update can be installed immediately.
0 of these updates are security updates.
To see these additional updates run: apt list --upgradable


Last login: Sun Nov 15 16:23:18 2020 from 192.168.0.13
```

Ilustración 48. Conexión con usuario ansible.

Antes de ejecutar el playbook de ansible, podemos probar con el módulo ping, para ello debemos tener un archivo llamado ansible.cfg (Por defecto ansible leerá desde /etc/ansible) que contendrá la ruta desde donde se cargará el inventario (ilustración 49 y 50).

```
➜ ansible -u ansible -m ping webservers
192.168.0.18 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

Ilustración 49. Verificación ansible.

```
Documents/TFM_Practice/ansible_tfm
➜ cat inventory/inventory.yml
[webservers]
192.168.0.18

[webservers:vars]
user=ansible
ansible_ssh_private_key_file= /home/ecastro/Documents/TFM_Practice/tfm
```

Ilustración 50. Configuración inventario.

La configuración de ansible contendrá tres partes, la primera será un módulo común para cualquier máquina que queramos incluir en la infraestructura, este contendrá 9 pasos a ejecutar, uno de ellos será la inclusión de un usuario sudo para demostrar el uso de la bóveda de ansible y no dejarlo en texto plano o con hashes en el sistema de control de versiones, para ello se creó una bóveda de ansible con el comando ansible-vault create roles/common/vaults/sudo, para la creación se solicita un password que debe almacenarse para poder editar la bóveda en caso de ser necesario. La estructura del mismo se definió como se muestra en la ilustración 51:

```
ssh_users_sudo:
  - { name: 'edison', uid: 2001 , groups: 'sudo', passwd: '$6$Z6jabs[redacted]
O4BLgfqqL6QL1', pub_key: 'ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAICzedlkJ0qCx1C0PsnO3y1uKzkUTFb4SsOYvGSz3Xe8F'}
```

Ilustración 51. Bóveda ansible.

El llamado se hace desde ansible incluyendo la bóveda, se crea el código para que se pueda extender a más usuarios en caso de ser necesario (Ilustración 52).

```
- name: Incluir la bóveda para la creación de usuarios
  include_vars:
    file: vaults/sudo

- name: Agregar los usuarios que tendrán uso de sudo
  user:
    name: "{{ item.name }}"
    password: "{{ item.passwd }}"
    uid: "{{ item.uid }}"
    shell: /bin/bash
    state: present
    groups: "{{ item.groups }}"
  loop: "{{ ssh_users_sudo }}"
  no_log: true

- name: Agregar las llaves para los usuarios
  authorized_key:
    user: "{{ item.name }}"
    state: present
    key: "{{ item.pub_key }}"
  loop: "{{ ssh_users_sudo }}"
  no_log: true
```

Ilustración 52. Creación de usuarios en ansible.

Si intentamos leer el archivo roles/common/vaults/sudo encontraremos una cadena propia de ansible (Ilustración 53).


```
$ANSIBLE_VAULT;1.1;AES256
35333335346634366333653966653031393739643133363832363365613738373734373231346164
6330616236333831616530633439663163623438666562320a35643065366132663032376166313 6
39633938393666333563613632383538653536656237643666336666643963353361613861363836
38663136636336376 50a31626635343437393464386432623461393138333616436396164653232 39
38633531626637613734393233303231313033356630376539383032366130306631383632363964
62316134653835396562363461303465386464363131303266663064343230636665363835633933
38303431643431623736613165626239613231303763366465663564343835306530333534636636
37353231663565313035306166353163323130646263313733303164333661343364633839353561
30316536626262383330663362623134666632643162326164663534326530373830393632353035
32363963636434373532643734613663623136616163303764303237316136356561333936363066
65616134383162363033383733656161383336643734663061356336386634366539326636346230
37323063373061316665373835346539336162383465363131393463323136643838633137653136
```


Ilustración 53. Lectura Bóveda con editor.

Paso siguiente se hace el llamado del playbook de ansible solo con el rol common obteniendo la salida de la ilustración 54:

```
➜ ansible-playbook -i inventory/inventory.yml server_2004_tfm.yml -u ansible

PLAY [webservers] ***********************************************************

TASK [Gathering Facts] ******************************************************
ok: [192.168.0.18]

TASK [common : Actualizar todos los paquetes a la última versión] ***********
ok: [192.168.0.18]

TASK [common : Instalar unattended-upgrades] ********************************
ok: [192.168.0.18]

TASK [common : Deshabilitar el módulo update_hostname en el cloud-init] *****
ok: [192.168.0.18]

TASK [common : Generar el archivo /etc/hosts] *******************************
changed: [192.168.0.18]

TASK [common : Deshabilitar IPv6 en caso de no estarlo utilizando] **********
changed: [192.168.0.18]

TASK [common : Configurar unattended-upgrades] ******************************
changed: [192.168.0.18]

TASK [common : Incluir la bóveda para la creación de usuarios] **************
ok: [192.168.0.18]

TASK [common : Agregar los usuarios que tendrán uso de sudo] ****************
changed: [192.168.0.18] => (item=None)
changed: [192.168.0.18]

TASK [common : Agregar las llaves para los usuarios] ************************
changed: [192.168.0.18] => (item=None)
changed: [192.168.0.18]

PLAY RECAP ******************************************************************
192.168.0.18               : ok=10   changed=5    unreachable=0    failed=0
```

Ilustración 54. Ejecución rol common ansible.

Para la utilización de slack necesitamos otro token, por motivos ilustrativos se muestra la forma de creación del mismo (ilustración 55):

```
➜ ansible-vault encrypt_string 'token_de_slack_aca' --name 'slack_token'
slack_token: !vault |
          $ANSIBLE_VAULT;1.1;AES256
          346130383166393739323962383236303730383263306366363232393335316131666165653166
          376433363537393765373438323962326264333262383734 0a353365626537363661633765313330
          333466663463393338363537316532663662383637316231653839663033346533373365363961 34
          663961393566653435 0a6131643866316238303737626138646530613335386166393032343635 61
          393739653063323930663664613466343663393861656239366166393831643334 35
Encryption successful
```

Ilustración 55. Creación de token en slack.

Este token será utilizado para realizar las alertas a un canal específico, la documentación específica para el manejo de estos tokens se puede encontrar en Slack (2020). El archivo con el llamado a todos los roles es el de la ilustración 56:

```
---
- hosts: webservers
  become: yes
  roles:
    - common
    - software
    - hardening
```

Ilustración 56. Archivo server_2004_tfm.yml.

La forma de ejecución del playbook de ansible es la mostrada en la ilustración 57:

```
→ ansible-playbook -i inventory/inventory.yml server_2004_tfm.yml -u ansible
```

Ilustración 57. Ejecución playbook ansible

Después de ejecutar todo el playbook hacemos una ejecución de lynis para verificar el estado de nuestro servidor, así como las recomendaciones del software, en este estado ya tendríamos el hardening realizado.

```
Lynis security scan details:

Hardening index : 85 [################   ]
Tests performed : 266
Plugins enabled : 2

Components:
- Firewall              [V]
- Malware scanner       [X]

Scan mode:
Normal [V]  Forensics [ ]  Integration [ ]  Pentest [ ]

Lynis modules:
- Compliance status     [?]
- Security audit        [V]
- Vulnerability scan    [V]

Files:
- Test and debug information      : /var/log/lynis.log
- Report data                     : /var/log/lynis-report.dat

================================================================================

Lynis 3.0.2

Auditing, system hardening, and compliance for UNIX-based systems
(Linux, macOS, BSD, and others)

2007-2020, CISOfy - https://cisofy.com/lynis/
Enterprise support available (compliance, plugins, interface and tools)

================================================================================
root@edison:/opt/lynis#
```

Ilustración 58. Resultado lynis

Un sistema de linux recién instalado tiene este valor en 55, un índice de 85 es aceptable para lo que estamos buscando, sin embargo, la herramienta da unas recomendaciones para aumentar el nivel de seguridad en caso de querer aumentar el valor obtenido (ilustración 59).

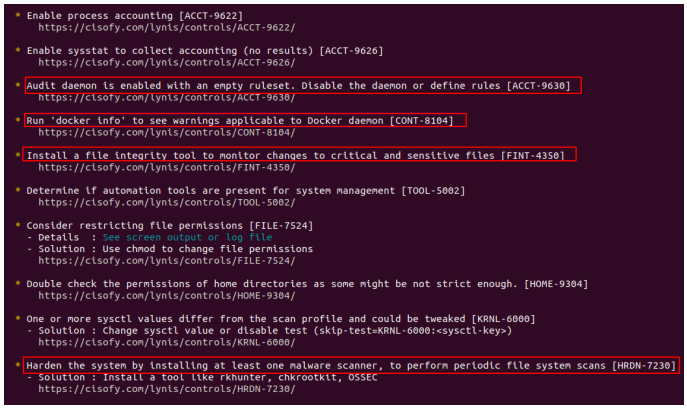

Ilustración 59. Sugerencias lynis

3.2.3 Terraform:

El proceso de instalación de terraform es similar al paso anterior, dirigirnos a la página oficial y descargar el binario, descomprimirlo en alguna carpeta de nuestro gusto y ponerlo disponible para ejecutarlo desde cualquier parte del sistema operativo, por motivos de compatibilidad con checkov debe ser la versión 0.12 o superior, al momento de escritura de este trabajo la última versión estable es la 0.13.5.

Uno de los puntos importantes en seguridad a tener en cuenta en terraform es el manejo del estado de los recursos creados, terraform basa gran parte de su lógica en estos archivos al punto que depende de esos archivos para ser idempotente, si no se hace configuración adicional por defecto se creará en la máquina desde la que se ejecute terraform con el nombre terraform.tfstate, si por ejemplo creo una máquina de base de datos, el usuario y el password serán almacenados en este archivo, motivo por el cual debemos encontrar una estrategia para protegerlo, es allí cuando debemos utilizar un concepto llamado backend, en nuestro caso haremos uso de un backend remoto en s3 aplicando el cifrado correspondiente para disminuir la superficie de ataque. Se debe utilizar un backend por cada ambiente, otra recomendación es no mantener el archivo de estado en el repositorio de código, pues se pueden tener secretos, no se puede hacer bloqueo entre diferentes personas trabajando en el mismo código o se puede olvidar subirlo y tener múltiples vistas de la infraestructura (Ilustración 60 y 61.

```
resource "aws_s3_bucket" "terraform_state" {
  bucket = "uoc-edisoncast-tfm-state"
  # El versionamiento es importante para saber los archivos que cambiamos
  versioning {
    enabled = true
  }

  # Debemos activar server side encryption por defecto
  server_side_encryption_configuration {
    rule {
      apply_server_side_encryption_by_default {
        sse_algorithm = "AES256"
      }
    }
  }
}
```

Ilustración 60. Creación de bucket en s3

```
# Debemos evitar que varias personas modifiquen el mismo archivo al mismo tiempo
# Se utiliza dynamo para almacenar las transacciones
#https://www.terraform.io/docs/providers/aws/r/dynamodb_table.html
# Usamos lockId
resource "aws_dynamodb_table" "terraform_locks" {
  name         = "uoc-edisoncast-tfm-terraform-lock"
  billing_mode = "PAY_PER_REQUEST"
  hash_key     = "LockID"
  attribute {
    name = "LockID"
    type = "S"
  }
}
```

Ilustración 61. Creación de tabla de dynamo

En este trabajo final de máster no tendremos bases de datos a desplegar, pero en caso tal de tenerlas debemos definir el proceso para el almacenamiento y la rotación de secretos, desde el 2014 es un issue abierto en terraform, si bien existen soluciones alternas, no se puede hacer de forma nativa aún (ilustración 62).

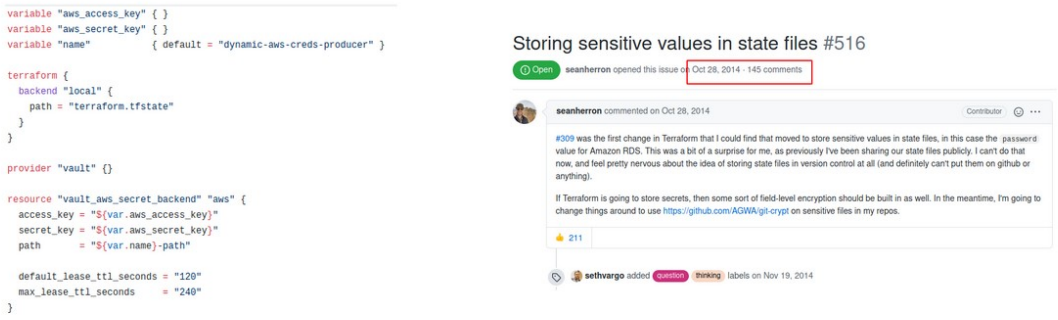

Ilustración 62. Secretos en terraform

Se puede minimizar el impacto utilizando vault o el gestor de secretos del proveedor utilizado, uno de los errores más comunes en infraestructura como código es generar

instancias de base de datos con el password en el código, por defecto este usuario tendrá permisos de root (Ilustración 63 y 64)

```
resource "aws_db_instance" "default" {
  allocated_storage    = 20
  storage_type         = "gp2"
  engine               = "mysql"
  engine_version       = "5.7"
  instance_class       = "db.t2.micro"
  name                 = "mydb"
  username             = "foo"
  password             = "foobarbaz"
  parameter_group_name = "default.mysql5.7"
}
```

Ilustración 63. Error seguridad bases de datos terraform

```
data "aws_secretsmanager_secret_version" "credentials" {
  secret_id = "[redacted]-creds"
}

locals {
  db_creds = jsondecode(
    data.aws_secretsmanager_secret_version.credentials.secret_string
  )
}

resource "aws_db_instance" "default" {
  allocated_storage    = 20
  storage_type         = "gp2"
  engine               = "mysql"
  engine_version       = "5.7"
  instance_class       = "db.t2.micro"
  name                 = "mydb"
  username             = local.db_creds.username
  password             = local.db_creds.password
  parameter_group_name = "default.mysql5.7"
}
```

Ilustración 64. Recomendación secretos terraform

Se crea un módulo para el despliegue de la máquina en EC2, paso siguiente se corre ansible para el endurecimiento del mismo, similar a como se realizó el proceso en local con vagrant.

3.2.4 Gitlab:

Para ejecutar el ambiente de integración continua necesitamos poner nuestro código en el sistema de control de versiones, para ello, es necesario abrir una cuenta en gitlab.com, además empezar el proceso de configuración de un archivo .gitlab-ci.yml en la raíz del proyecto (ver ilustración 65).

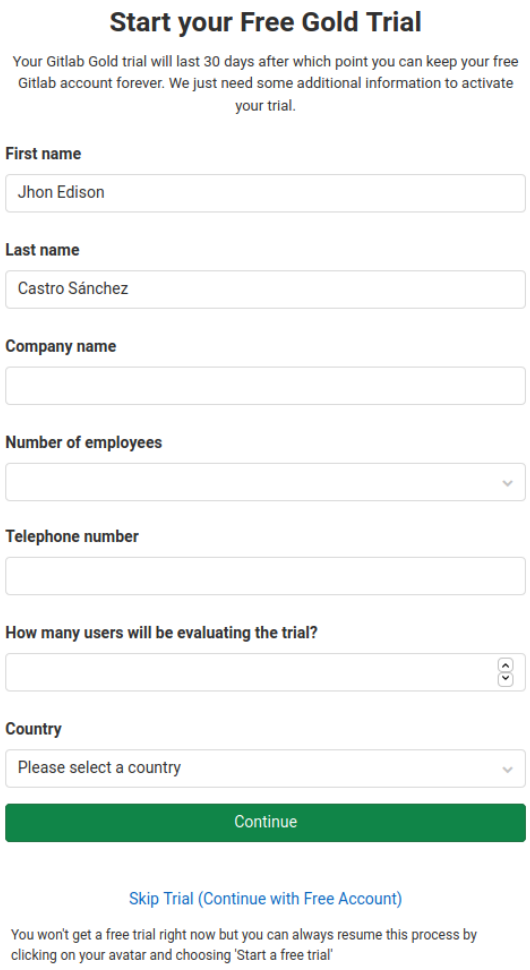

Ilustración 65. Registro gitlab

Después de realizar el registro, se crea el archivo necesario para correr las pruebas de ci, en este caso gitlab utilizará algo llamado gitlab-runners para hacer cada uno de los pasos definidos (ver ilustración 66).

En la parte inicial se ejecutan dos trabajos, el primero es un formato para terraform y el segundo es checkov para validación de la infraestructura de terraform, si bien, solamente tenemos tres recursos ya tenemos algunas consideraciones a tener en cuenta (ver ilustración 67,68, 69).

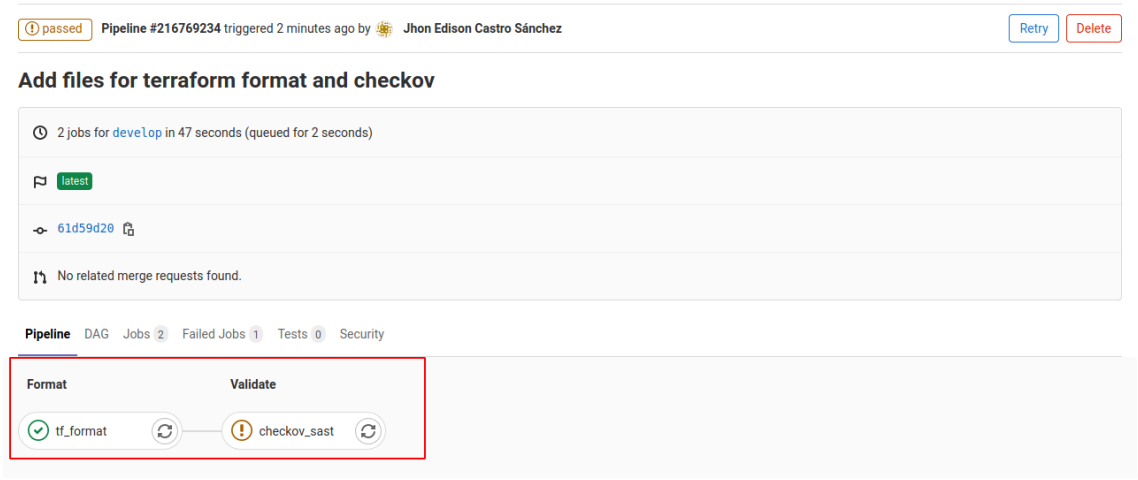

Ilustración 66. Validación de infraestructura en gitlab

```
checkov_sast:
  image:
    name: bridgecrew/checkov
    entrypoint:
      - '/usr/bin/env'
      - 'PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin'
  stage: validate
  script:
    - apt update && apt install -y jq
    - checkov -o json -d . | jq -r '.results.failed_checks[] | "\(.check_name) \(.file_path)"' | tee gl-sast-checkov.json
  allow_failure: true
  artifacts:
    name: gl-sast-checkov.json
    paths:
      - gl-sast-checkov.json
    reports:
        sast: gl-sast-checkov.json
    expire_in: 1 week
```

Ilustración 67. Definición de checkov en Gitlab-ci

```
$ checkov -o json -d . | jq -r '.results.failed_checks[] | "\(.check_name) \(.file_path)"' | tee gl-sast-checkov.json
Ensure Instance Metadata Service Version 1 is not enabled /terraform/modulos/computacion/EC2/main.tf
Ensure every security groups rule has a description /terraform/modulos/computacion/EC2/main.tf
Ensure the S3 bucket has access logging enabled /terraform/ambiente/objetos/s3/main.tf
Ensure S3 bucket has MFA delete enabled /terraform/ambiente/objetos/s3/main.tf
Ensure Dynamodb point in time recovery (backup) is enabled /terraform/ambiente/objetos/s3/main.tf
```

Ilustración 68. Recomendaciones de checkov

Hasta este momento solo se hace revisión del formato de terraform, lo ideal es hacer pruebas a la creación de la infraestructura con terraform kitchen o con terratest, pruebas que se salen del alcance del trabajo final de máster.

El objetivo es entonces simular un entorno productivo, para esto necesitamos una aplicación, motivo por el cual desarrollé una aplicación en nodejs extremadamente sencilla con algunas líbrerías en el package.json, la idea es hacer análisis de secretos,

análisis de composición de software, pruebas unitarias, creación de una imagen docker, SAST, escáneo de la imagen construida.

Para la verificación de docker utilicé hadolint, al ejecutarlo me mostró dos pasos a corregir, los cuales fueron agregados en un nuevo commit.

```
$ hadolint Dockerfile
Dockerfile:9 DL3020 Use COPY instead of ADD for files and folders
Dockerfile:13 DL3020 Use COPY instead of ADD for files and folders
```

Ilustración 69*.* Hadolint sugerencia

4. Solución final DevSecOPs

El link del código se puede encontrar en https://gitlab.com/jhon_edison_castro_sanchez_uoc/tfm/-/tree/develop

El pipeline completo se ve en la (ilustración 70)

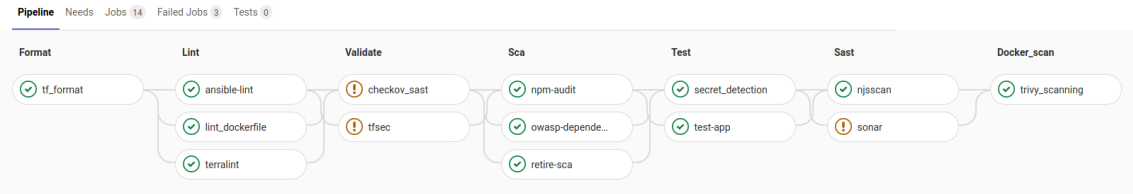

Ilustración 70. Pipeline completo

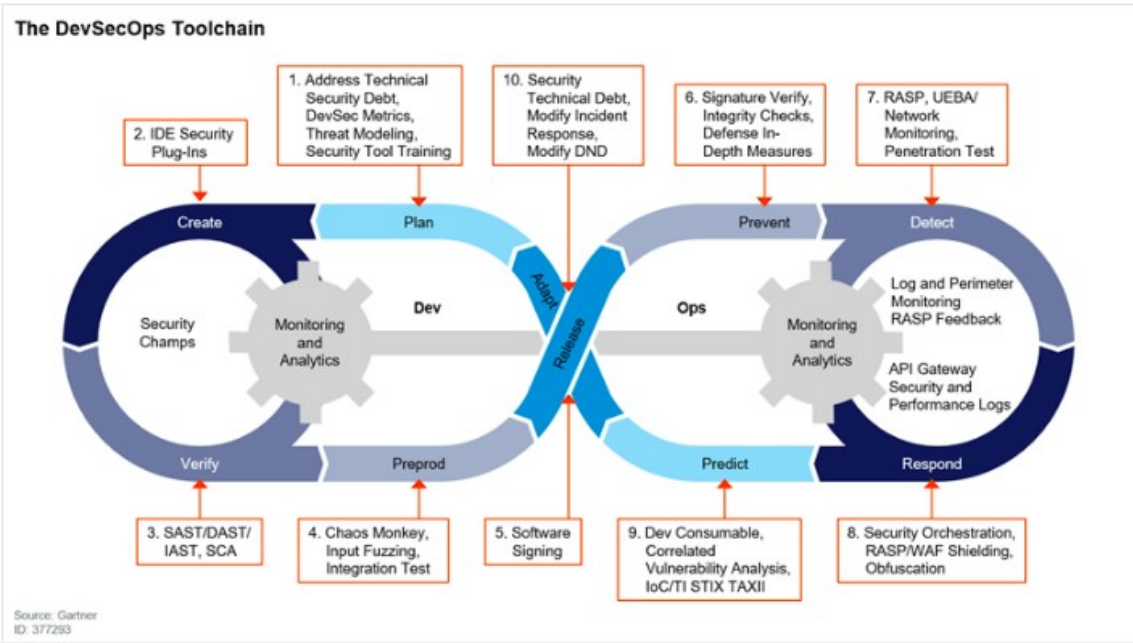

Ilustración 71. DevSecOpsToolchain

Si se hace la revisión del DevSecOps toolchain establecido en Gartner (Ilustración 71) frente al pipeline construido, se logra identificar que cumplimos con algunos puntos mencionados, sin embargo, existen otros que no fueron contemplados en la definición de este proyecto. Si bien existen recomendaciones generales cada caso de uso debe trabajarse de forma independiente y ojalá siguiendo un proceso establecido. El pipeline desarrollado cumple con el principio de estar completamente automatizado y donde se puede aplicar el concepto de despliegue continúo, con esto logramos securizar nuestra aplicación en diferentes fases utilizando para ello diferentes herramientas.

4.1 Resultados obtenidos:

Cada que se realice un push al repositorio se ejecutará el pipeline construido, actualmente se ejecuta todo el pipeline en cada una de las ramas que se tengan en el repositorio.

4.1.1 Análisis definición de infraestructura:

Se obtienen algunas recomendaciones de validación de infraestructura tanto en checkov como en tfsec.(Ver ilustración 72 y 73).

```
Ensure Instance Metadata Service Version 1 is not enabled /terraform/modulos/computacion/EC2/main.tf
Ensure all data stored in the Launch configuration EBS is securely encrypted /terraform/modulos/computacion/EC2/main.tf
Ensure all data stored in the Launch configuration EBS is securely encrypted /terraform/modulos/computacion/EC2/main.tf
Ensure the S3 bucket has access logging enabled /terraform/ambiente/objetos/s3/main.tf
Ensure Dynamodb point in time recovery (backup) is enabled /terraform/ambiente/objetos/s3/main.tf
```

Ilustración 72. Recomendaciones de checkov

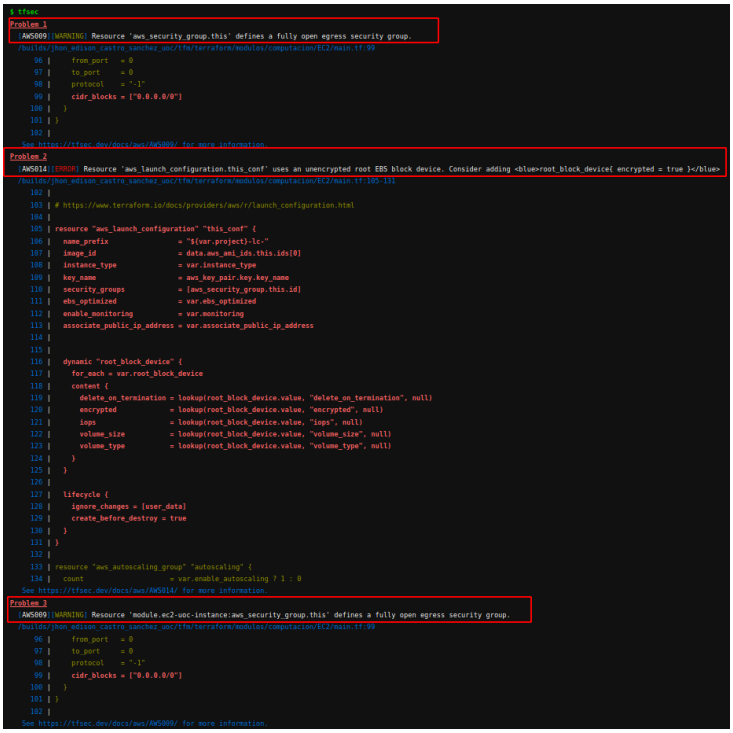

Ilustración 73. Recomendaciones tfsec

4.1.2 Análisis de SCA:

En SCA, se utilizó npm audit, herramienta para verificar que los paquetes utilizados no cuenten con vulnerabilidades conocidas, también se ejecuta owasp dependency check y retire.js. (Ver ilustración 74)

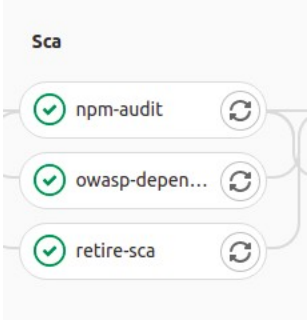

Ilustración 74. Análisis de dependencias

4.1.3 Análisis de secretos:

Para la detección de secretos se utiliza la capa gratuita de gitlab que detecta si se agregan secretos en el código como lo son claves, contraseñas y tokens de API. Para realizar este proceso gitlab utiliza dos herramientas open source como lo son gitleaks y TruffleHog. Se recomienda además que este proceso sea incluido en un pre-commit para así evitar que los secretos vayan al repositorio central.

4.1.3 SAST:

En el apartado de SAST se utilizó nodejsscan y SonarQube en un gitlab-cloud runner, sin embargo, la extensión para ejecutarlo en un servidor de nuestra elección es muy sencilla, se incluye el archivo de docker-compose para la ejecución de sonarQube en una máquina diferente (Ilustración 74)

```
version: '3.7'

services:
  sonarqube:
    image: sonarqube
    expose:
      - 9000
    ports:
      - "127.0.0.1:9000:9000"
    networks:
      - sonarnet
    environment:
      - SONARQUBE_JDBC_URL=jdbc:postgresql://db:5432/sonar
      - SONARQUBE_JDBC_USERNAME=sonar
      - SONARQUBE_JDBC_PASSWORD=sonar
    volumes:
      - sonarqube_conf:/opt/sonarqube/conf
      - sonarqube_data:/opt/sonarqube/data
      - sonarqube_extensions:/opt/sonarqube/extensions
      - sonarqube_bundled-plugins:/opt/sonarqube/lib/bundled-plugins

  db:
    image: postgres
    networks:
      - sonarnet
    environment:
      - POSTGRES_USER=sonar
      - POSTGRES_PASSWORD=sonar
    volumes:
      - postgresql:/var/lib/postgresql
      - postgresql_data:/var/lib/postgresql/data

networks:
  sonarnet:

volumes:
  sonarqube_conf:
  sonarqube_data:
  sonarqube_extensions:
  sonarqube_bundled-plugins:
  postgresql:
  postgresql_data:
```

Ilustración 75. Docker-compose sonarQube

El proceso para ejecutar el archivo es docker-compose up -d (ilustración 76), lo que se realizará será bajar las dos imágenes (sonarqube y postgres) desde docker hub, compartir la red sonarnet, y poner el servicio en el puerto 9000. Cabe aclarar que sonarQube antes de la versión 8.6 tiene vulnerabilidades en su api, específicamente, un atacante puede lograr omitir la autenticación por medio de sonar scanner CVE-2020-28002 (Gutierrez, s.f.).

El cambio que debemos realizar en el archivo gitlab.yml es utilizar un tag que puede ser definido como servidor_sonar y en el servidor hacer el proceso de instalación del gitlab-runner y la ejecución de la receta de docker compose que se habló en el punto anterior.

```
stage: SAST
variables:
  SONAR_URL: http://localhost:9000
  SONAR_BASELINE: BASELINE
  SONAR_PROJECT: $CI_PROJECT_PATH_SLUG
script:
  - |
    sonar-scanner -Dsonar.qualitygate.wait=true \
      -Dsonar.projectVersion=$SONAR_BASELINE \
      -Dsonar.projectKey=$SONAR_PROJECT \
      -Dsonar.projectName=$SONAR_PROJECT \
      -Dsonar.sources=. \
      -Dsonar.host.url=$SONAR_URL \
      -Dsonar.login=$SONAR_ID \
      -Dsonar.branch.name=$CI_COMMIT_REF_NAME \
      -Dsonar.sourceEncoding=utf-8
tags:
  - servidor_sonar
```

Ilustración 76. Creación de sonar en gitlab runner propio

4.1.4 Análisis de contenedores:

Para este proceso se utilizó trivy buscando por vulnerabilidades de tipo HIGH y critical. Al realizar el análisis se encuentran dos de tipo HIGH referentes a openssl. (Ver ilustración 77).

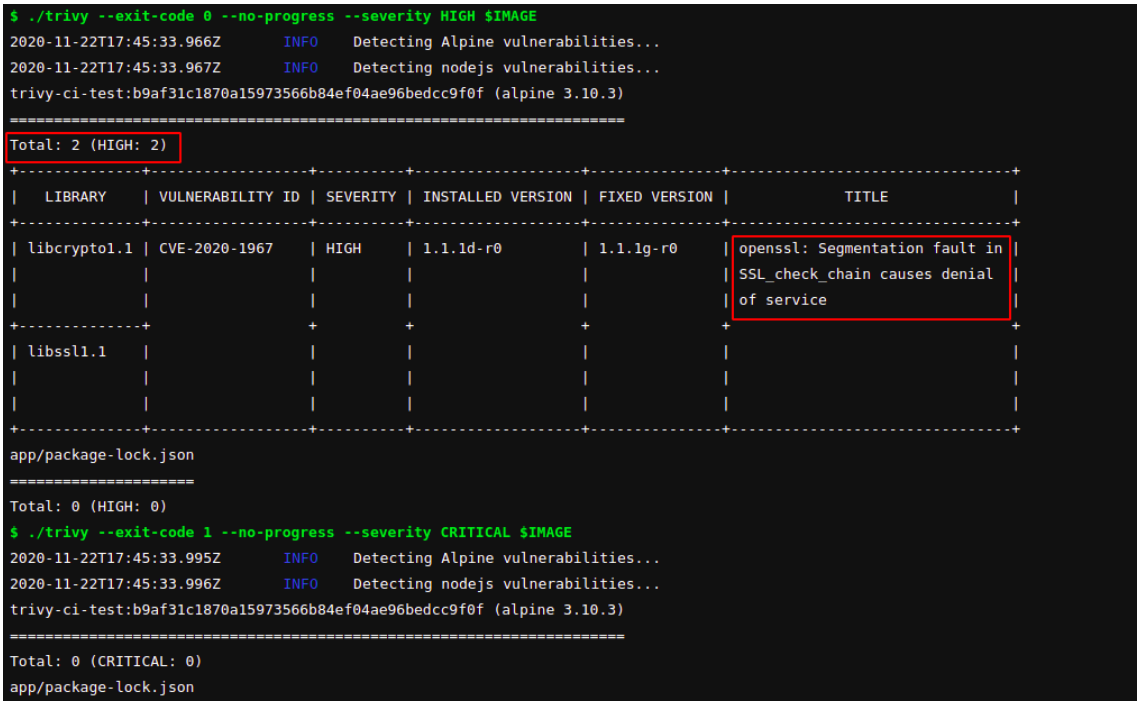

Ilustración 77. Análisis de imagen docker

Una de las recomendaciones de seguridad para aplicar en contenedores, es intentar utilizar el concepto de distroless para evitar algunas capas, sin embargo, desde la experiencia del autor, se han encontrado algunos bloqueos con este enfoque como por ejemplo, la necesidad de alguna librería en el sistema operativo que es requerida por el lenguaje de programación.

Existen temas adicionales como el análisis DAST con Owasp ZAP, revisión de licenciamiento, políticas como código y la integración de cada uno de los artefactos generados por el pipeline en una herramienta central como lo puede ser defect dojo. Con estos se tendría una vista desde diferentes capas, la idea es poder alinearlo a la estrategia empresarial para capacitación del equipo de desarrollo y operaciones en estos temas, también se podría tener alertas por cada regla incumplida o hacer la generar un issue para la gestión inmediata de incidentes, similar al proceso que se realiza para las alertas del endurecimiento de servidores.

5. Conclusiones

1. Las herramientas open source permiten obtener un pipeline completo de seguridad en cada una de las capas, modelado de amenazas, análisis de secretos, SCA, SAST, DAST, tener una visión clara del código fuente ejecutado, ayudando a la colaboración entre la comunidad y minimización de costos. También, la creación, gestión, mantenimiento y endurecimiento de los servidores a través de un proceso de definición de imágenes endurecidas tanto para sistemas operativos como para contenedores.
2. En el primer estudio de DevSecOps Latam que contó con la participación del autor en calidad de squad líder del capítulo Colombia, se encontraron varios puntos claves: a partir del diligenciamiento de la encuesta de DevSecOps LATAM por parte de 136 personas (siendo el segundo país con más participantes), a pesar de no ser una muestra significativa para el país, logró evidenciar que las prácticas de DevOps están madurando rápidamente, pero la seguridad no se está automatizando en fases tempranas del ciclo para desarrollo de software al no considerarse un punto crítico, aunque se cuenten con avances en estudios mundiales, constituyen para Latinoamérica un largo camino largo por recorrer.
3. Esta investigación permitió conocer la existencia de pocas herramientas para validación de las reglas de seguridad aplicadas a la infraestructura como código, el cual, cada día tiene mayor presencia en el mercado, generando errores comunes como puertos abiertos, no cifrados de la información o poco nivel de log para realización de procesos de auditoría.
4. El uso de Docker, los modelos de nube actuales y las nuevas herramientas generan una falsa confianza de seguridad y no aseguran el software contra riesgos.
5. La creación casi que inmediata de servidores debe estar acompañada de una política de endurecimiento de los sistemas y un plan de seguridad alineado a los objetivos empresariales.
6. Se pudo identificar que las imágenes de docker más utilizadas tienen vulnerabilidades y pueden ser aprovechadas en algún momento en el tiempo si no se define un plan de actualización, esto da una oportunidad para ejecutar controles y políticas de higiene en seguridad informática.
7. A medida que se incremente la cantidad de componentes de software de código libre en la base de código de los productos, es necesario tener una gestión e inventario de dependencias en la organización, como política interna que incluya nuevas librerías al código fuente y una revisión periódica de las librerías utilizadas.
8. La unión de cada una de las salidas del proceso del pipeline contribuye a una mejor gestión de la inclusión de seguridad al proceso DevOps, mediante el uso de herramientas ASTO se puede tener soporte de diferentes fuentes de datos, alertas a las personas implicadas ante nuevos sucesos, y gestión de incidentes compartidos por cada área de la organización.
9. Existe una evolución de diferentes herramientas en le mercado para la aplicación de DevSecOps como lo son Azure, https://sken.ai/, Gitlab. Estas incorporan SAST, DAST, SCA, análisis de secretos, creación de issues automáticos, alertas, dashboards todo en un modelo SaaS.

10. Existen nuevas vertientes a revisar desde la academia como lo son el enfoque no ops, revisión de seguridad en aplicaciones serverless, manejo y gestión de secretos con herramientas como VAULT o los propios de los proveedores de nube como secrets manager y parameter store en AWS. Conceptos como codeql y sem-grep serán fundamentales para mejorar la seguridad de las aplicaciones en un futuro cercano.

6. Glosario

AST: Application Security Testing, técnicas de prueba depara probar la seguridad identificando vulnerabilidades o agujeros de seguridad en las aplicaciones.

ASTO: Application Security Testing Orchestration

CI: Integración continua, traducida desde el inglés como continuous integration.

CD: Traducida desde el inglés como integración continua o despliegue continuo (Continuos delivery, continuos deployment)

DAST: o prueba de seguridad de aplicación dinámica. Ayuda a hallar las vulnerabilidades y debilidades en la seguridad de una aplicación en ejecución, normalmente aplicaciones web.

IAST: Pruebas de seguridad de aplicaciones interactivas. Combina las técnicas de análisis estáticos y dinámicos (SAST + DAST) creando un análisis global completo del sistema.

IaC: Infraestructura como código.

GC: Gestión de la configuración

Literatura gris: “Conjunto de documentos, de muy diversa tipología, que no son editados o que se publican pero distribuyen a través de canales poco convencionales (tesis doctorales, actas de congresos, informes de investigación, memorias, proyectos, patentes, normas, traducciones científicas, etc.), por lo que suelen plantear problemas especiales para conocerlos y localizarlos” , se refiere a contenidos producidos por profesionales en base a sus experiencias prácticas [13, 15]

Manifiesto: Escrito en que se hace pública declaración de doctrinas, propósitos o programas.

MLR (Multivocal literature review) : Es una forma de revisión sistemática de la literatura (SLR) con la inclusión de literatura gris.

NIST: Instituto nacional de estándares y tecnología.

NCP: Definido como el Programa Nacional de Lista de Verificación (NCP), fundado por el NIST SP 800-70, es el depósito del gobierno de los EE. UU. De listas de verificación de seguridad (o puntos de referencia) disponibles públicamente que brindan una guía detallada de bajo nivel sobre cómo establecer la configuración de seguridad de los sistemas operativos y aplicaciones.

Open Scap: Open security content automation protocol

Pipeline: Consiste en evadir los desechos del proceso para programación de software, y se emplea para proporcionar reportes rápidos al equipo durante la ejecución.
Resiliencia: Construir ambientes u organizaciones que son tolerantes a cambios e incidentes.

RASP: Protección de seguridad de la aplicación en tiempo de ejecución. Así mismo como IAST, RASP o Runtime Application Security Protection, se ejecuta en la aplicación.

SAST: o prueba de seguridad de aplicación estática. Facilita a los programadores hallar las vulnerabilidades de seguridad en el código fuente de la aplicación.

SCA: Software Composition Analysis identica, monitorea y crea un inventario de todos los componentes de terceros en el software para el responder a las vulnerabilidades en ellos.

TDD: Test driven development, es utilizado para generar las pruebas en etapas más tempranas del desarrollo de software.

TTR Medio: El tiempo medio que se tarda en corregir los defectos descubiertos en una aplicación.

7. Bibliografía

Atlassian, (2020). What is devops? https://www.atlassian.com/devops*.*

Balaji, S. & Murugaiyan, M. S. (2012). Waterfall vs. v-model vs. agile: A comparative study on sdlc, International Journal of Information Technology and Business Management, 2, pp. 26–30

Bass, L., Holz, R., Rimba, P., Tran, A. B. & Zhu,L. (2015). Securing a Deployment Pipeline. 2015 IEEE/ACM 3rd International Workshop on Release Engineering (4 − 7). Florence, Italy, 2015.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Suther land, J. & Thomas, D. (2001). Manifesto for agile software development https://agilemanifesto.org/. Accedido en línea 04.10.2020

Burgess, M. & Couch, A. (2006). Modeling Next Generation Configuration Management Tools. 131-147.

Canonical. (2020). Wiki development code names https://wiki.ubuntu.com/DevelopmentCodeNames

Bridgecrew. (2020). Prevent cloud misconfigurations during build time https://www.checkov.io/, https://github.com/bridgecrewio/checkov

Cappelli, D. M., Moore, A. P., & Trzeciak, R. F. (2006). Insider Threats in the SDLC: Lessons Learned From Actual Incidents of Fraud, Theft of Sensitive Information, and IT Sabotage. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.

Carvalho, J. A. Moreira, L. & Rosso,L. C. (Sin Fecha). Introducción a DevOps y DevSecOps Un guía para entender la revolución en IT. Buenos Aires, Argentina. https://cloudlegion.com.ar/files/Introduccion%20a%20Devops%20y%20DevSecOps.pdf

Carter, K. (2017). Francois raynaud on devsecops, IEEE Software, 34, p. 93–96.

Carvalho, J. A., Moreira, L., Rosso, L. & Ibiri, C.(2020). Infraestructuras ágiles el corazón de DevOps en IT OPS https://docs.google.com/viewer?url=https://devsecops-latam.org/contenidos/infraestructuras-agiles-el-corazon-de-devops.pdf

Carvalho, J. A., Moreira, L. Rosso, L. & Ibiri, C. (2020). DevSecOps security testing automation https://docs.google.com/viewer?url=https://devsecops-latam.org/contenidos/devsecops-ast.pdf&embedded=true

Cois, C. A. Yankel, J. & Connell, A. (2014). "Modern DevOps: Optimizing software development through effective system interactions," 2014 IEEE International Professional Communication Conference (IPCC), Pittsburgh, PA, pp. 1-7, doi: 10.1109/IPCC.2014.7020388.

Chen, L. (2015). Continuous delivery: Huge benefits, but challenges too, IEEE Software, 32, pp. 50–54.

Willis, J. (2012). The convergence of DevOps. https://itrevolution.com/the-convergence-of-devops/

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

Delgado, O., & Pablo., J. (2015). Análisis de seguridad y calidad de aplicaciones (Sonarqube).

DevOps Research & Assessment (DORA). (2020). Accelerate state of DevOps 2019 .https://services.google.com/fh/files/misc/state-of-devops-2019.pdf

Duvall, P., Matyas, S., & Glover, A. (2007). Continuous Integration: Improving Software Quality and Reducing Risk (The Addison-Wesley Signature Series).

Eng, C. (2019). State of sotware security Veracode. Retrieved 20 December 2020,from https://www.veracode.com/sites/default/files/pdf/resources/sossreports/state-of-software-security-volume-9-veracode-report.pdf

Fagan, M. E. (1976). Design and code inspections to reduce errors in program development. IBM.

Fitzgerald, B. & Stol, K.-J. (2015). Continuous software engineering: A roadmap and agenda, Journal of Systems and Software, 123, pp. 176–189. DOI: 10.1016/j.jss.2015.06.063

Fowler, M. (2020). bliki: ContinuousIntegrationCertification. Retrieved 21 December 2020, from https://martinfowler.com/bliki/ContinuousIntegrationCertification.html

Gallego, G. (2020). DevOps al estilo Amazon. Retrieved 20 December 2020, from https://www.youtube.com/watch?v=Xh2YpJmNXucJabbari, R., bin Ali, N., Petersen, K., & Tanveer, B. (2016, May). What is DevOps? A systematic mapping study on definitions and practices. In Proceedings of the Scientific Workshop Proceedings of XP2016 (pp. 1-11).

Gartner, Inc. (2019). The Secret to DevOps success. https://www.gartner.com/smarterwithgartner/the-secret-to-devops-success/

Haber, W. (2020). Top 6 security trends in GitLab-hosted projects https://about.gitlab.com/blog/2020/04/02/security-trends-in-gitlab-hosted-projects/

Henke, M. (2020). Why Happy Developers Create More Secure Code by DJ Schleen & Derek Weeks. Retrieved 20 December 2020, from https://www.alldaydevops.com/blog/why-happy-developers-create-more-secure-code-by-dj-schleen-derek-weeks

Horvath, M., Zumerle, D. & Gardner, D. (2020). Magic Quadrant for Application Security Testing.

Humble, J. & Farley. D. (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (1st. ed.). Addison-Wesley Professional.

Humble, J. & Molesky, J. (2020). Why enterprises must adopt devops to enable continuous delivery, Cutter IT Journal, https://www.cutter.com/sites/default/files/itjournal/fulltext/2011/08/itj1108.pdf. p. 6 − 12*.*

Jarret, T., Wysopal, C. & Eng, C. (2020). State of sotware security Veracode volumen 10. Retrieved 20 December 2020, from https://www.veracode.com/sites/default/files/pdf/resources/sossreports/state-of-software-security-volume-10-veracode-report.pdf

Koskinen, A. (2019). DevSecOps: building security into the core of DevOps.

Lee, J. S. (2018). The devsecops and agency theory, in 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), IEEE, 2018, pp. 243–244

Lietz, S., Shrikant, R. et al (2020). DevSecOps Manifesto, https://www.devsecops.org/

Loughman, K. (2019). The devops model: Rapid software delivery and incident management. https://victorops.com/blog/the-devops-model-rapid-software-delivery-and-incident-management,

Mansfield-Devine, S. (2018). DevOps: finding room for security. Network Security, 2018(7), 15-20.

Michener, J. R. & Clager, A. T. (2016). Mitigating an oxymoron: Compliance in a devops environments, in 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), vol. 1, IEEE, 2016, p. 396–398.

Miller, A. & Zitzman, S. (2020). State of open source security report https://f.hubspotusercontent10.net/hubfs/1699665/sooss_2020_final_v2.pdf

Miller, A. (2018). Security automation with ansible. https://www.ansible.com/hubfs/2018_Content/AnsibleAutomates-AnsibleForSecurityAutomation.pdf?hsLang=en-us

Moreira, L. & Ibiri, C. (2020). DevSec Opss!!! Los casos de no éxito de DevSecOps . https://www.youtube.com/watch?v=5HSscsplJjM*,*

Myrbakken, H. & Colomo-Palacios, R. (2017). DevSecOps: A Multivocal Literature Review. International Conference on Software Process Improvement

National Vulnerability Database (2020). National Vulnerability Database Retrieved 21 December 2020, from https://www.cisecurity.org/, https://www.open-scap.org/

Neto GTG, Santos WB, Endo PT & Fagundes RAA. (2019). Multivocal literature reviews in software engineering: Preliminary findings from a tertiary study. ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), Empirical Software Engineering and Measurement (ESEM), ACM/IEEE International Symposium on. September 2019:1-6. doi:10.1109/ESEM.2019.8870142

Null, C. (2020).10 companies killing it at devops. https://techbeacon.com/devops/ 10-companies-killing-it-devops.

Palo alto networks, (2020). Unit 42 Cloud Threat https://www.paloaltonetworks.com/resources/research/digital-executive-summary-unit-42-cloud-threat-report-spring-2020

Poppendieck, M. & Poppendieck, T. (2003). Lean Software Development: An Agile Toolkit: An Agile Toolkit, Addison-Wesley.

Raynaud, F. (2020). DevSecOps business benefits and best practices - SD Magazine.Retrieved 20 December 2020, from https://sd-magazine.com/securite-numerique-cybersecurite/devsecops-business-benefits-and-best-practices

Redhat (2020). Redhat, DevSecOps y la seguridad de DevOps. Retrieved 20 December 2020, from https://www.redhat.com/es/topics/devops/what-is-devsecops

Stroud, R. (2020). 2018: The Year Of Enterprise DevOps. Retrieved 20 December 2020, from https://go.forrester.com/blogs/2018-the-year-of-enterprise-devops/

Ur Rahman, A. A., & Williams, L. (2016). Security practices in DevOps. In Proceedings of the Symposium and Bootcamp on the Science of Security (pp. 109-111).

Schlossnagle, T. (2018). Monitoring in a devops world, Comunicaciones ACM, 61, pp. 58–61.

Schleen, D. & Weeks, D. E. (2020). DevSecOps Community Survey. https://www.sonatype.com/2020-dso-community-surveys, 2020.

Swartout, P., 2012. Continuous Delivery and DevOps: a QuickStart Guide. Packt Publishing.

Synopsys. (2020). Open source security and risk analysis report, https://www.synopsys.com/software-integrity/resources/analyst-reports/2020-open-source-security-risk-analysis.html

Tamburri, D. A., Di Nucci, D., Di Giacomo, L. & Palomba, F. (2018). Omniscient devops analytics, in International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment, Springer, 2018, p. 48–59.

Willis. J. (2010). What DevOps Means To Me. Chef.io 16 Julio de 2010. https://blog.chef.io/2010/07/16/what-devops-means-to-me

Federal trade commission. (2018). Federal trade commission. Vol. 83, No. 80 https://www.ftc.gov/system/files/documents/federal_register_notices/2018/04/152_3054_uber_revised_consent_analysis_pub_frn.pdf

Meli, M., McNiece, M.R., & Reaves, B. (2019). How Bad Can It Git? Characterizing Secret Leakage in Public GitHub Repositories. NDSS.

SAFECode security engineering training. (2020). System hardening 101. https://safecode.org/lessons/system-hardening-101/

Zimmermann, M., Staicu, C., Tenny, C., & Pradel, M. (2019). Small World with High Risks: A Study of Security Threats in the npm Ecosystem. USENIX Security Symposium.

Cohn, M. (2009). *Succeeding with Agile: Software Development Using Scrum* (1st. ed.). Addison-Wesley Professional.

Mburano, B. & Weisheng, S. (2018). Evaluation of Web Vulnerability Scanners Based on OWASP Benchmark. 1-6. 10.1109/ICSENG.2018.8638176.

Haber, W. (2020). GitLab's DevSecOps Methodology Assessment https://about.gitlab.com/resources/devsecops-methodology-assessment/.

Vagrant (Sin fecha). Hashicorp, Development environments made easy https://www.vagrantup.com/

Slack. (2020). Cómo crear y volver a generar tokens de API https://slack.com/intl/es-co/help/articles/215770388-C%C3%B3mo-crear-y-volver-a-generar-tokens-de-API

Netflix technology blog. (2010). 5 Lessons We've Learned Using AWS, https://netflixtechblog.com/5-lessons-weve-learned-using-aws-1f2a28588e4c,

Gutierrez, C. (Sin fecha) SonarQube – Auditando al Auditor – Parte I https://csl.com.co/sonarqube-auditando-al-auditor-parte-i/

Zhou, X. (2020, June). How to Treat the Use of Grey Literature in Software Engineering. In Proceedings of the International Conference on Software and System Processes (pp. 189-192).

¿Qué es la entrega continua? – Amazon Web Services. (2020). Retrieved 21 December 2020, from https://aws.amazon.com/es/devops/continuous-delivery/