



Introducción a DevSecOps para la mejora de los
procesos de desarrollo de software con
herramientas Open Source

Profesor:

Joaquín López Sánchez-
Montañés

Fecha: 18-Junio-2023

Autor:

Alberto Élez Villamarín



Esta obra está sujeta a una licencia de Reconocimiento-
NoComercial-SinObraDerivada [3.0 España de Creative Commons](https://creativecommons.org/licenses/by-nc-nd/3.0/es/)

FICHA DEL TRABAJO FINAL

Título del trabajo:	<i>Introducción a DevSecOps para la mejora de los procesos de desarrollo de software con herramientas Open Source</i>
Nombre del autor:	<i>Alberto Élez Villamarín</i>
Nombre del profesor/a:	<i>Joaquín López Sánchez-Montañés</i>
Fecha de entrega:	06/2023
Titulación:	<i>Grado de Ingeniería Informática</i>
Área del Trabajo Final:	<i>Administración de redes y sistemas operativos</i>
Palabras clave	<i>DevSecOps, SecDevOps, DevOps. gestión, proyectos, metodologías, agile, desarrollo, operaciones, Open Source</i>
Resumen del Trabajo:	
<p>Este documento describe los trabajos realizados en el ámbito del Trabajo Fin de Grado (TFG). Se detallarán las tareas a ejecutar y la documentación generada a lo largo del proyecto.</p> <p>El propósito principal de este TFG es describir de una manera clara en qué consiste y cómo implementar DevSecOps en una organización. Se tratará de explicar de manera detallada el concepto de DevSecOps, de identificar los beneficios de su implementación en las empresas, los desafíos a los que se enfrentan las organizaciones durante su proceso de adopción, así como analizar su ciclo de vida.</p> <p>También se analizará el estado del arte actual respecto a las herramientas que se utilizan para trabajar con DevSecOps y se desarrollará un ejemplo práctico de implementación de DevSecOps con herramientas software Open Source.</p>	
Abstract:	
<p>This document describes the work carried out in the scope of the Degree's Thesis (TFG). The tasks performed will be detailed and the documentation generated throughout the process.</p> <p>The main purpose of this TFG is to describe in a clear way what is DevSecOps and how it works in an organization. It will try to explain in detail the concept of DevSecOps, to identify the benefits of implementing it in companies, the challenges that organizations face during their adoption process, as well as analyzing their life cycle.</p> <p>The current state of the art will also be analyzed with respect to the tools used to work with DevSecOps and a practical example of DevSecOps implementation with Open Source software tools will be developed.</p>	

Dedicatoria y agradecimientos

Dedico este trabajo final de carrera a mis padres, las mejores personas que conozco. Son el espejo en el que me gustaría reflejarme.

También a mis suegros, que con su infatigable ayuda y cuidado de sus nietos hacen posible que yo pueda seguir estudiando.

Se lo dedico también a mi mujer Silvia, mi punto de apoyo en esta vida, sin ella a mi lado no podría vivir, y a mis hijos Alberto y Marcos, que son los dos soles que iluminan mi vida.

A todos ellos les pido perdón por las ausencias, por todo el tiempo que le he tenido que dedicar a este proyecto y no a ellos.

Quisiera darle las gracias a mi consultor, Joaquín López Sánchez-Montañés, por todo lo que me ha ayudado y todo lo que me ha apoyado durante la realización de este TFG.

Índice

1	Introducción	9
1.1	Descripción del proyecto.....	9
1.2	Justificación del proyecto.....	9
1.3	Motivación para la realización el proyecto.....	9
1.4	Ámbito de aplicación del proyecto.....	10
1.5	Objetivos.....	10
2	Planificación.....	11
2.1	Hitos principales del TFG.....	11
2.2	Descripción de los hitos principales	11
2.3	Productos obtenidos.....	12
2.4	Desglose de las tareas del TFG.....	12
2.5	Enfoque y método seguido	12
3	Introducción a DevOps	14
3.1	¿Qué es DevOps?.....	14
3.2	Principios de DevOps.....	16
3.3	Objetivos de DevOps.....	19
3.4	El ciclo de vida de DevOps	19
3.4.1	Ciclo de vida de las metodologías ágiles.....	19
3.4.2	El ciclo de vida DevOps.....	20
3.5	Estado del arte de las herramientas DevOps	21
3.5.1	Gestión de la configuración del código fuente	22
3.5.2	Integración Continua (CI).....	22
3.5.3	Testing	23
3.5.4	Entrega Continua y Despliegue Continuo.....	24
3.5.5	Monitorización	25
3.5.6	Alertas.....	26
3.5.7	Seguridad.....	27
3.5.8	Virtualización y containerización	27
3.5.9	Orquestación	29
3.5.10	Comunicación y colaboración	30
4	Introducción a DevSecOps.....	32
4.1	¿Qué es DevSecOps?.....	32
4.2	DevSecOps vs SecDevOps.....	34
4.3	Ciclo de vida de desarrollo seguro (SSDLC)	34
4.4	Herramientas de pruebas de seguridad en software	36
4.5	Modelado de amenazas	36
4.6	Análisis estático de la seguridad del código (SAST).....	39
4.7	Análisis de secretos.....	41
4.8	Análisis de componentes del software (SCA).....	42
4.9	Análisis dinámico de la seguridad de las aplicaciones (DAST)	44
4.10	Análisis de seguridad de los contenedores	45
4.11	Análisis de la infraestructura como código (IAC)	46
4.12	Gestión de vulnerabilidades.....	48

5	Aplicación práctica de implementación de DevSecOps.....	50
5.1	DevSecOps Maturity Model	50
5.2	Pipelines de Jenkins para DevSecOps	51
5.3	Ejecución del Pipeline de Jenkins y análisis de resultados	54
5.3.1	Análisis de los resultados de las pruebas estáticas (SAST).....	59
5.3.2	Análisis de los resultados de las pruebas dinámicas (DAST).....	65
6	Conclusiones.....	73
6.1	Conclusiones y lecciones aprendidas	73
6.2	Objetivos alcanzados.....	73
6.3	Líneas de trabajo futuras	74
7	Glosario de términos.....	75
8	Bibliografía	76
9	Anexos.....	78
9.1	Instalación de las herramientas DevSecOps	78
9.1.1	Instalación de Java	78
9.1.2	Instalación de Docker	79
9.1.3	Instalación de Docker Compose.....	81
9.1.4	Instalación de Jenkins.....	82
9.1.5	Instalación de SonarQube.....	86
9.1.6	Instalación de Retire.JS	91
9.1.7	Instalación de OWASP Dependency-Check	92
9.1.8	Instalación de NodeJSScan	92
9.1.9	Instalación de Nuclei.....	93
9.1.10	Instalación de Wapiti.....	94
9.1.11	Instalación de ZAP Proxy.....	95
9.1.12	Instalación de Trivy.....	98

ÍNDICE DE TABLAS

TABLA 1. HITOS PRINCIPALES DEL TFG.....	11
TABLA 2. HERRAMIENTAS OPEN SOURCE PARA LA GESTIÓN DE LA CONFIGURACIÓN DEL CÓDIGO FUENTE (SCM)	22
TABLA 3. HERRAMIENTAS OPEN SOURCE PARA LA INTEGRACIÓN CONTINUA (CI)	23
TABLA 4. HERRAMIENTAS OPEN SOURCE PARA LA ENTREGA Y EL DESPLIEGUE CONTINUO (CD)	25
TABLA 5. HERRAMIENTAS OPEN SOURCE PARA LA MONITORIZACIÓN.....	26
TABLA 6. HERRAMIENTAS OPEN SOURCE PARA LA GENERACIÓN DE ALERTAS	27
TABLA 7. HERRAMIENTAS OPEN SOURCE PARA LA VIRTUALIZACIÓN	29
TABLA 8. HERRAMIENTAS OPEN SOURCE PARA LA CONTAINERIZACIÓN	29
TABLA 9. HERRAMIENTAS OPEN SOURCE PARA LA ORQUESTACIÓN.....	30
TABLA 10. HERRAMIENTAS OPEN SOURCE PARA LA COMUNICACIÓN Y LA COLABORACIÓN	31
TABLA 11. HERRAMIENTAS OPEN SOURCE PARA EL THREAT MODELING.....	38
TABLA 12. HERRAMIENTAS OPEN SOURCE PARA SAST.....	40
TABLA 13. HERRAMIENTAS OPEN SOURCE PARA EL ANÁLISIS DE COBERTURA DE CÓDIGO	41
TABLA 14. HERRAMIENTAS OPEN SOURCE PARA EL ANÁLISIS DE SECRETOS.....	42
TABLA 15. HERRAMIENTAS OPEN SOURCE PARA SCA.....	43
TABLA 16. HERRAMIENTAS OPEN SOURCE PARA DAST	45
TABLA 17. HERRAMIENTAS OPEN SOURCE PARA EL ANÁLISIS DE CONTENEDORES.....	46
TABLA 18. HERRAMIENTAS OPEN SOURCE PARA EL ANÁLISIS DE LA IAC	47
TABLA 19. HERRAMIENTAS OPEN SOURCE PARA EL ANÁLISIS DE LA COMPLIANCE AS CODE	48
TABLA 20. HERRAMIENTAS OPEN SOURCE PARA LA GESTIÓN DE VULNERABILIDADES.....	49
TABLA 21. GLOSARIO	75

ÍNDICE DE FIGURAS

FIGURA 1. PLANIFICACIÓN GENERAL DE LOS HITOS PRINCIPALES DEL TFG	12
FIGURA 2. PLANIFICACIÓN DE LAS TAREAS DEL TFG	12
FIGURA 3. HISTORIA DE DEVOPS.	14
FIGURA 4. TRIÁNGULO DEVOPS.	15
FIGURA 5. STATE OF DEVOPS REPORT 2018	16
FIGURA 6. MODELO DEVOPS.	17
FIGURA 7. MURO DE CONFUSIÓN.	17
FIGURA 8. PERSONA CON PERFIL EN FORMA DE T.	18
FIGURA 9. MARCO DE TRABAJO DE SCRUM.	19
FIGURA 10. CICLO DE VIDA DE DEVOPS.	20
FIGURA 11. MEJORA CONTINUA Y RESOLUCIÓN DE PROBLEMAS	21
FIGURA 12. TABLA PERIÓDICA DE LAS HERRAMIENTAS DEVOPS.	21
FIGURA 13. MÁQUINAS VIRTUALES Vs CONTENEDORES.	28
FIGURA 14: DEVOPS VS DEVSECOPS.	32
FIGURA 15. OPEN SOURCE CODEBASE VULNERABILITIES	33
FIGURA 16. PRUEBAS EN CADA FASE DEL SSDLC.	35
FIGURA 17: PIRÁMIDE AST	36
FIGURA 18. DSOMM LEVELS	50
FIGURA 19. VULNADO ARCHITECTURE	52
FIGURA 20. OWASP JUICE SHOP	52
FIGURA 21. TOP 10 WEB APPLICATION SECURITY RISKS 2021	53
FIGURA 22. TRIVY	99

1 Introducción

1.1 Descripción del proyecto

Actualmente, muchas empresas de desarrollo de software están implementando algún tipo de metodología ágil con el fin de generar software más frecuentemente a través de pequeños incrementos. Sin embargo, eso no implica que el producto esté listo para entrar en producción y para entregar al cliente, ya que muchas veces se descuida la calidad y la seguridad del software generado. También suelen aparecer problemas de ejecución en el entorno de producción, generando a veces conflictos entre el equipo de Desarrollo y el de Operaciones.

Para solucionar estos problemas surgió el movimiento o la cultura “DevOps”, que se basa en una relación de trabajo colaborativo entre el equipo de Desarrollo y el equipo de Operaciones de TI, lo que posibilita un rápido flujo de trabajo (esto es, altas tasas de despliegue), al tiempo que aumenta la confiabilidad, la estabilidad, la resistencia y la seguridad del entorno de producción.

Sin embargo, muchas empresas piensan que la gestión de la seguridad es un paso al final de ciclo del desarrollo de software (SDLC, Software Development Life Cycle) y que está reservada a los expertos de seguridad. Esto implica que, si se detectan errores en la fase de despliegue, se deben corregir y después volver a generar el software desde el principio, incrementando por tanto los costes del proyecto y probablemente incurriendo en retrasos en la entrega del mismo.

Para mitigar estos problemas surgió “DevSecOps”, que incorpora pruebas de seguridad automáticas e integradas en todas las fases del ciclo de vida del software, denominándose Secure Software Development Life Cycle (SSDLC), que nos permite generar software que sea seguro en cada una de las fases del proyecto, así como conocer sus riesgos y poder gestionarlos desde el principio. De esta manera los equipos de Desarrollo y Operaciones deben trabajar de manera conjunta con el equipo de Seguridad durante todas las fases del proyecto.

1.2 Justificación del proyecto

Aunque la cultura DevOps está actualmente ampliamente adoptada en las empresas que desarrollan software, todavía no existe una cultura de aplicar la seguridad desde el inicio de un proyecto y, por tanto, no se ha estandarizado la implementación de DevSecOps. Esa falta de adopción de DevSecOps, no se debe a que los arquitectos y los desarrolladores se opongan o no entiendan cómo funciona, sino que en muchos casos, la razón se debe a que los stakeholders de los proyectos, en su mayoría consideran los presupuestos y las ganancias a corto plazo en lugar de a largo plazo, ya que, implementar un pipeline completo de DevSecOps lleva tiempo y muchos stakeholders prefieren centrarse en las funcionalidades de la aplicación en vez de asegurarse de su seguridad.

La idea de este proyecto es ofrecer una visión global de DevSecOps, mostrando cuales son los beneficios que aporta a la empresa su adopción, así como analizar el software actual disponible para implementar el ciclo completo del SSDLC aplicando los principios de DevSecOps.

También se va a generar un ejemplo sencillo de implementación, que pueda servir de guía o de punto de apoyo a todo aquél que lo lea para iniciar la implantación de DevSecOps en sus propios proyectos.

1.3 Motivación para la realización el proyecto

Los motivos para llevar a cabo este TFG son estudiar y conocer a fondo DevSecOps, así como analizar las herramientas empleadas actualmente para cubrir todo el ciclo de vida SSDLC, haciendo hincapié en las herramientas Open Source.

Otro de los motivos principales es aprender y desarrollar un ejemplo de pipeline de Jenkins, utilizando herramientas Open Source, de manera que se cubra el ciclo de vida completo SSDLC y que pueda ser utilizado por cualquiera que lea este TFG en sus proyectos de manera casi inmediata, customizándolo a partir de estos ejemplos según sus necesidades.

1.4 **Ámbito de aplicación del proyecto**

La implantación de DevSecOps se puede (y se debe) llevar a cabo en la mayoría de las empresas de desarrollo de software, sobre todo en aquellas donde ya estén utilizando DevOps.

El cada vez mayor número de ciber ataques que se están produciendo actualmente implica que la gestión de la seguridad es una parte imprescindible de todo el software que se genera y que los costes de la no-seguridad son mucho más elevados que los costes de una implementación adecuada de la seguridad.

1.5 **Objetivos**

El primer objetivo que se pretende alcanzar con la realización de este TFG es dar una visión completa de la cultura o el movimiento DevOps: porqué surgió la necesidad de implantarlo en las empresas, sus principios, sus roles, su ciclo de vida, sus ventajas e inconvenientes, su relación con las metodologías ágiles, etc.

Otro objetivo es analizar el modelo DevSecOps, porqué es necesario implementarlo desde las fases iniciales del proyecto, incluyendo procedimientos de automatización y análisis de código (estático y dinámico) y estableciendo un ciclo de vida de desarrollo de software seguro (SSDLC).

Otro de los objetivos principales es analizar el estado el arte actual de las herramientas de software empleadas en DevOps y en DevSecOps, dando una visión de las más importantes para cada una de las áreas involucradas en todas las fases de los proyectos de desarrollo de software.

El último de los objetivos principales y el más importante, es desarrollar un ejemplo de implantación de un proyecto de desarrollo de software cubriendo todo el ciclo de vida DevSecOps (SSDLC), empleando herramientas Open Source, de manera que cualquier empresa pueda utilizarlo, adaptándolo a sus necesidades, y alcance de esa manera un DevSecOps Maturity Model Level 1.

2 Planificación

2.1 Hitos principales del TFG

Los hitos de entrega del proyecto están fijados por el plan docente de la asignatura desde el inicio del curso y se descomponen en cuatro entregas. Además, hay otros dos hitos importantes que son la defensa del TFG ante el Tribunal y la publicación de las notas finales.

Las fechas de entrega de los hitos son las siguientes:

HITO	Nombre	FECHA	ESTADO
1.1	PEC1 - Propuesta de plan de trabajo TFG.	19/03/2023	Finalizado
1.2	PEC2 - Entre el 40% y el 60% de todo el TFG.	23/04/2023	Finalizado
1.3	PEC3 - Entre el 80% y el 90% de todo el TFG.	28/05/2023	Finalizado
1.4	PEC4 - Entrega Final.	18/06/2023	Finalizado
1.5	Defensa del TFG ante el Tribunal.	TBD	Pendiente
1.6	Publicación de las notas finales.	TBD	Pendiente

Tabla 1. Hitos principales del TFG

2.2 Descripción de los hitos principales

Los hitos principales del TFG son los siguientes:

- **Hito 1.1:** Este hito establece el comienzo del desarrollo del TFG. En él se desarrollará el documento actual y se corresponde con la entrega de la PEC1. En este documento se propondrá una planificación inicial que será la guía a seguir para el desarrollo del TFG.
- **Hito 1.2:** En este hito se comienza a desarrollar la primera parte de la memoria del TFG, donde se deberá completar entre el 40% y el 60% del trabajo del TFG. Se corresponde con la entrega de la PEC2.
- **Hito 1.3:** En este hito se comienza a desarrollar la segunda parte de la memoria del TFG, donde se deberá completar entre el 80% y el 90% del trabajo del TFG. Se corresponde con la entrega de la PEC3.
- **Hito 1.4:** En este hito se entrega la memoria final del TFG, junto con una presentación resumen y un video para la defensa del TFG. Este hito se corresponde con la PEC4.
- **Hito 1.5:** En este hito se inicia el debate con el tribunal, que revisará todo el trabajo realizado y preguntará lo que estime conveniente.
- **Hito 1.6:** En este hito, el tribunal asignará una nota final al proyecto y concluirá el TFG.

Para tener una visión más esquemática de los hitos propuestos y del contenido general entregable en cada uno de ellos, se muestra a continuación un diagrama de Gantt en el que se pueden ver los hitos establecidos, sus nombres, el período de duración en días laborables y las fechas de inicio y finalización de cada uno de ellos.

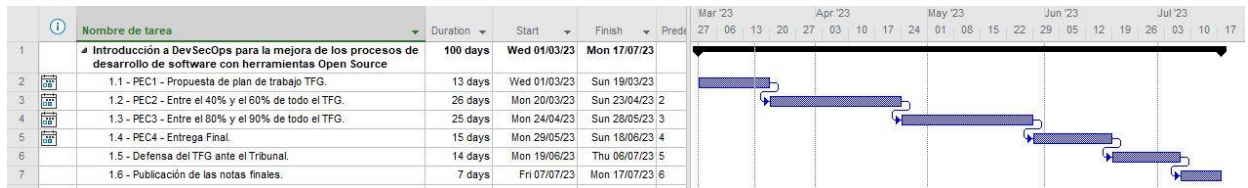


Figura 1. Planificación general de los Hitos principales del TFG

2.3 Productos obtenidos

Al finalizar el curso, tras realizar las tareas descritas en la planificación, se deberán entregar los siguientes productos:

- **Memoria:** documento con el informe final que recoge todo el trabajo realizado en el TFG.
- **Presentación ejecutiva:** presentación multimedia con un resumen del trabajo realizado en el TFG junto con un vídeo con los comentarios correspondientes a cada diapositiva, con una duración máxima de 20 minutos.
- **Código de ejemplo:** se entregará el código fuente de los scripts de configuración del pipeline de Jenkins empleados en la realización del TFG para que puedan servir de punto de inicio para las personas que lean este TFG y deseen generar su propio pipeline DevSecOps.

2.4 Desglose de las tareas del TFG

En cada periodo de tiempo asociado a un hito de entrega descrito en el punto 2.1, se han establecido unas tareas a realizar. En este apartado se muestran las subtareas a realizar asociadas a cada uno de esos periodos, de manera que se pueda tener una visión más detallada y estructurada del conjunto total del trabajo que hay que desarrollar en cada periodo y para cada hito de entrega.

En el siguiente diagrama de Gantt se puede ver con detalle la planificación de las tareas de este TFG:

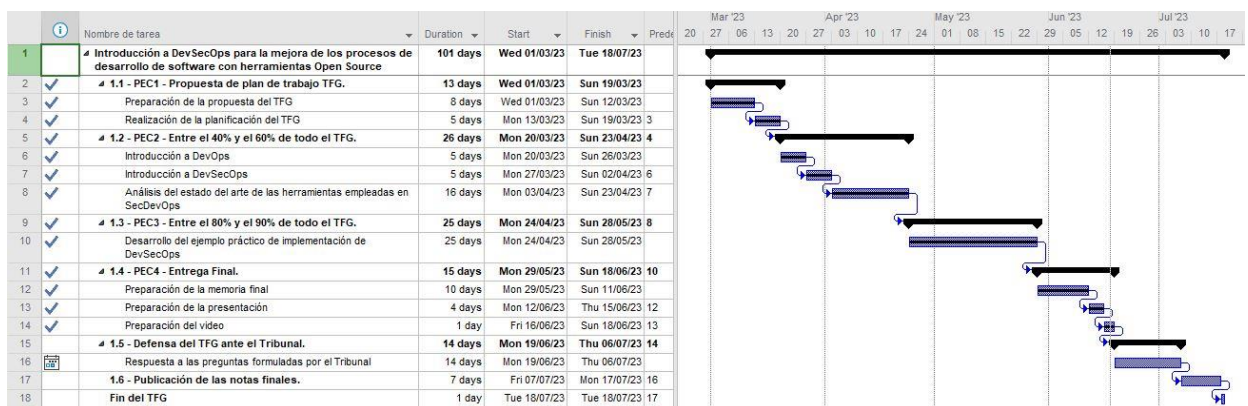


Figura 2. Planificación de las tareas del TFG

2.5 Enfoque y método seguido

El enfoque seguido para la realización del TFG es el de ir cumpliendo todos los objetivos del proyecto y en el orden establecido en el capítulo 2.4 de este documento, donde se detalla su planificación.

Para llevar a cabo la planificación, que se deberá cumplir estrictamente para poder terminar el proyecto en la fecha planificada, se dividirá el trabajo total del proyecto en tareas y éstas, a su vez, en subtareas.

- Se empezará describiendo que es DevOps, su historia, sus principios, sus objetivos principales, su ciclo de vida y su relación con las metodologías ágiles.
- A continuación, se analizará el estado el arte actual de las herramientas de software empleadas en DevOps, dando una visión de las herramientas Open Source más importantes para cada una de las áreas involucradas en todas las fases de los proyectos de desarrollo de software.
- A continuación, se describirá que es DevSecOps, porqué es necesario adoptarlo desde el principio del desarrollo de un software y se explicará el ciclo de vida de desarrollo seguro (SSDLC).
- Después se analizará el estado el arte actual de las herramientas de software empleadas en DevSecOps, dando una visión de las herramientas Open Source más importantes para algunas de las áreas más relevantes del desarrollo de software seguro.
- Para terminar, se desarrollará un ejemplo de implantación de un pipeline en Jenkins, de un proyecto de desarrollo de software cubriendo todo el ciclo de vida DevSecOps, empleando herramientas Open Source.

3 Introducción a DevOps

3.1 ¿Qué es DevOps?

El término DevOps surge de la combinación de las palabras “desarrollo” (Development) y “operaciones” (Operations), ya que se trata de un movimiento profesional y cultural que promueve el trabajo colaborativo, la comunicación y la integración, de las personas que trabajan en el departamento de Desarrollo y las que trabajan en el departamento de Operaciones, para potenciar la entrega rápida, más eficiente, de aplicaciones y servicios, con una alta calidad y que además sean seguros y fáciles de mantener.

DevOps es un movimiento originado en 2007 por Patrick Debois, que en octubre de 2009 creó el evento “DevOpsDays” en Bélgica. En la siguiente imagen se puede ver un resumen de los hechos más relevantes de su evolución.

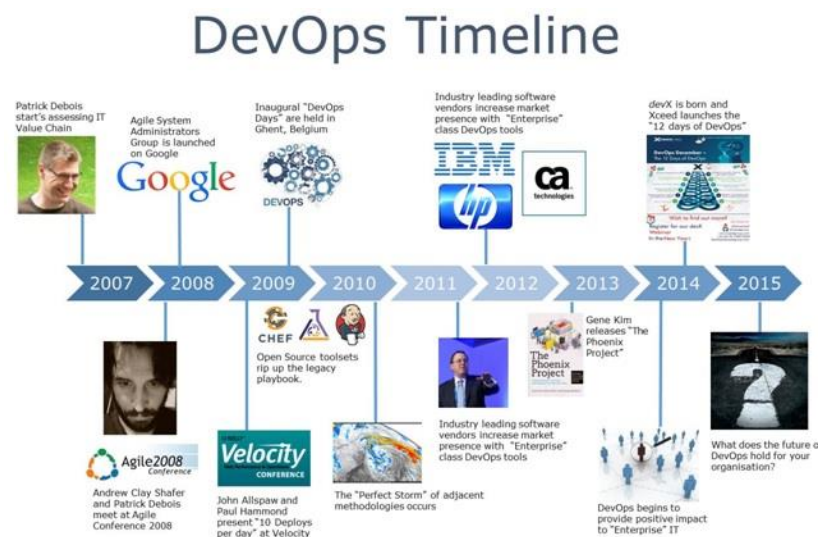


Figura 3. Historia de DevOps.

Fuente: <https://danielvillahermosa.wordpress.com/2019/11/27/la-evolucion-de-devops/>

Se puede definir DevOps como un conjunto de principios y prácticas que optimizan el tiempo de entrega de un producto software, gestionan la infraestructura como código y mejoran la experiencia del usuario en base a la retroalimentación de todo el proceso. DevOps engloba nuevos procesos, nuevas herramientas y nuevas formas de pensar.

La adopción de DevOps implica una nueva forma de trabajar, que trata de que la colaboración impere en el desarrollo, el funcionamiento y las áreas de negocio. Persigue conseguir la eliminación de los silos entre los distintos departamentos (sobre todo entre los de Desarrollo y Operaciones), la participación de las partes interesadas pertinentes y la distribución ágil de resultados de negocio automatizados. Se trata de hacer las cosas mejor con lo que hay disponible, añadir la tecnología apropiada donde el proceso necesita apoyo y conseguir una visibilidad total entre todas las partes implicadas.

En las organizaciones tradicionales, donde las barreras entre los distintos departamentos, con sus correspondientes directores, están muy bien definidas y cada departamento se preocupa únicamente de resolver los problemas que le afectan directamente y se echa la culpa de los posibles errores al resto de departamentos, esta nueva forma de trabajar implica cambios muy importantes, ya que las barreras entre los departamentos se difuminan (por ejemplo se busca formar grupos que asuman funciones de desarrollo y operativas a la vez) y todos se ven involucrados en la generación, de manera continua y en tiempos cortos, del producto y en dar visibilidad total de los procesos a toda la organización.

Cada grupo de trabajo debe romper las barreras existentes entre los departamentos de la organización permitiendo que un producto pase rápidamente desde el departamento de investigación al de diseño, luego al de desarrollo + producción, luego al de test + calidad y por último, a ventas, y con las necesidades de herramientas que permitan desplegar todas estas actividades en cada una de las fases y llevar el control de todos por los responsables de cada uno de los ámbitos, incluidos los funcionales y los de la organización (jefatura y directivos).

En la siguiente imagen se puede apreciar como DevOps se encuentra en la intersección de los equipos de Desarrollo, Operaciones TI y Calidad (QA):

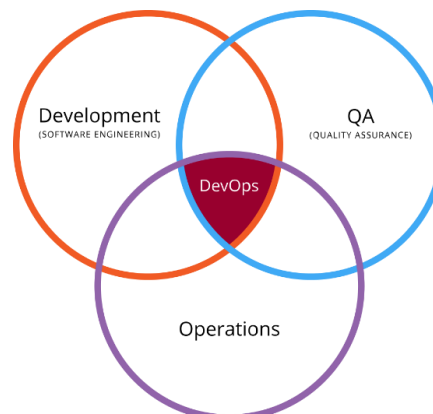


Figura 4. Triángulo DevOps.

Fuente: Fuente: <https://es.wikipedia.org/wiki/DevOps>

DevOps pretende facilitar el desarrollo continuo, la integración continua, la entrega continua y los procesos de monitoreo continuo con el fin de liberar el código más rápido (también reducir los tiempos de paso a producción y de entrega al cliente) y más a menudo para ayudar a la organización a responder de manera más ágil y eficiente a los cambios en los requisitos del negocio.

Los principales motivos estratégicos por los que una organización debe considerar implementar un modelo DevOps son los siguientes:

- **Reducción de costes:** al implementar DevOps se mejoran los procesos del negocio y se reducen por tanto sus costes.
- **Mejora de la calidad:** al estar continuamente generando versiones y probándolas se consigue que el producto final tenga una mejor calidad y se dé un mayor valor al cliente.
- **Mejora de la productividad:** al implementar ciclos más cortos de desarrollo con sus correspondientes pruebas en el entorno de producción se consigue aumentar la productividad y reducir los tiempos de entrega finales.
- **Diferenciación:** al implementar DevOps se reducen las ventajas competitivas de los competidores.
- **Innovación:** al implementar DevOps se consigue innovar introduciendo cambios radicales sobre los procesos del negocio, reduciendo los costes y mejorando la calidad, la eficiencia y el servicio al consumidor (menor time to market).

El objetivo principal de DevOps es acelerar la entrega de valor al cliente.

Los laboratorios Puppet (del 2013 al 2018) y desde entonces DORA de Google, generan un informe cada año llamado "*State of DevOps Report*". En este informe, se puede ver cómo las organizaciones de TI de alto rendimiento que usan los principios de DevOps lo están haciendo mucho mejor que sus competidores. El informe del año 2014 indicó que esas organizaciones de TI de alto rendimiento, implementaban 30 veces más frecuentemente, tenían plazos de entrega 200 veces más cortos, tenían 60 veces menos errores y también se recuperaban 16 veces más rápido. La siguiente imagen muestra los datos del 2018 y se puede apreciar una mejoría considerable, debido a que actualmente la mayoría de las empresas de desarrollo de software ya han adoptado los principios de DevOps:

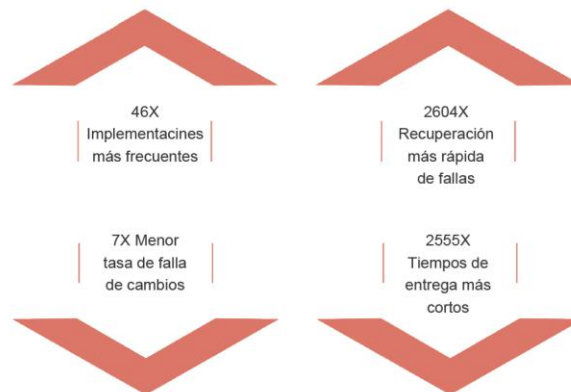


Figura 5. State of DevOps Report 2018

Fuente: <https://cloud.google.com/devops/state-of-devops?hl=es>

3.2 Principios de DevOps

DevOps no tiene un manifiesto similar al “manifiesto ágil” adoptado en las distintas metodologías ágiles, aunque en cierta manera también se encuentra incluido implícitamente en él.

Gene Kim en sus libros “DevOps Handbook” y “The Phoenix Project” describe los pilares fundamentales de DevOps como las tres vías (“Three Ways”):

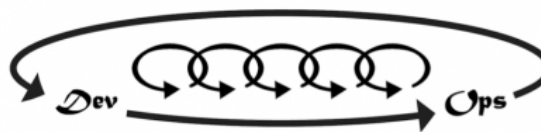
- **La primera vía, el Pensamiento sistémico:** se centra en todos los flujos que aportan valor para la entrega al cliente, del sistema completo, no de un solo departamento. Se basa en incrementar valor hasta el final sin dejar que los errores avancen hacia la derecha.



- **La segunda vía, Amplificar los bucles de retroalimentación:** consiste en crear bucles de retroalimentación de derecha a izquierda. Esto es, comprender y responder a las necesidades de los clientes.



- **La tercera vía, Cultura de experimentación y aprendizaje continuos:** consiste en experimentar y asumir riesgos de manera continuada, aprendiendo tanto de los éxitos como de los fracasos. Es necesario para mejorar en el trabajo diario.



DevOps se apoya sobre las siguientes metodologías y modelos:

- **Metodologías ágiles:** permiten incrementar la velocidad de desarrollo de los sistemas y mejorar el Time To Market.

- **Integración continua (Continuous Integration, CI):** consiste en integrar los cambios en el código fuente frecuentemente, con compilaciones automáticas, análisis estático y de cobertura del código y pruebas automáticas, para detectar errores de integración tan pronto como sea posible. Si alguno de estos pasos falla se da por erróneo el cambio y se notifica.
- **Entrega continua (Continuous Delivery, CD):** consiste en automatizar el paso del software a los entornos productivos. Todos los cambios se prueban y envían a un entorno de pruebas y/o producción. La configuración de la infraestructura como código nos permite crear plantillas de servicios complejos de forma fácil y poderlas almacenar bajo control de versiones como el resto del código fuente.
- **Despliegue continuo (Continuous Deployment):** se diferencia de la entrega continua en que no es necesario una aprobación explícita ni ninguna intervención manual para poder entregar una reléase a producción (se realiza automáticamente).

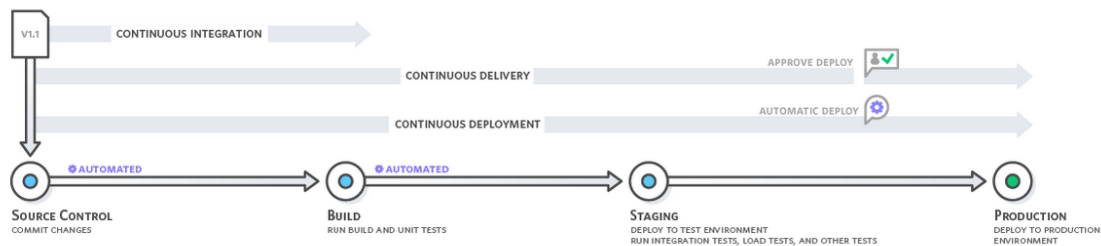


Figura 6. Modelo DevOps.

Fuente: <https://aws.amazon.com/es/devops/continuous-integration/>

Uno de los problemas que soluciona el movimiento DevOps es el llamado “muro de confusión”. Es el conflicto que surge cuando el equipo de Desarrollo hace el traspaso de código al equipo de Operaciones, y éste trabaja en el testing y en los scripts para el pase a producción. Es entonces cuando suelen surgir errores que no se sabe bien a quién atribuir, si a Desarrollo por entregar código con fallos o a Operaciones por realizar una mala configuración.

Cada departamento tiene sus propias prioridades, su forma de trabajar y vela por sus intereses: Desarrollo busca la manera de añadir nuevas funcionalidades y Operaciones busca la estabilidad por encima de todo. Al aparecer los errores surgen las fricciones y la falta de confianza entre los equipos. DevOps pretende aunar los intereses de ambos departamentos y crear un ambiente de confianza y cooperación mediante prácticas que fomentan una actitud positiva respecto al fallo y una cultura sin culpas (blameless culture), colaborativa, en la que la información fluya y el equipo se sienta comprometido con su trabajo.



Figura 7. Muro de confusión.

Fuente: <https://www.devopsagileskills.org/>

El modelo **CALMS**, acrónimo de Culture, Lean, Automation, Metrics y Sharing, persigue conseguir el cambio de mentalidad necesario en los equipos de Desarrollo y Operaciones para adoptar la cultura DevOps:

- **Cultura:** cambiar la manera de pensar y de comportamiento dentro de la organización. Desaparecen los equipos y se fomenta la cooperación.
- **Automatizar:** automatizar todo lo posible durante todo el ciclo, desde desarrollo hasta producción (CI/CD).
- **Lean:** enfocado en entregar valor al cliente y reduciendo lo que no aporta (los desperdicios).
- **Medir:** monitorizar y medir continuamente. Mostrar las mejoras.
- **Compartir:** compartir conocimiento y colaborar con el resto de trabajadores.

DevOps se fundamenta en los siguientes conceptos principales:

- **Mejora Continua:** DevOps defiende la mejora continua para minimizar el desperdicio.
- **Automatizar todo lo posible:** la automatización es un principio central del proceso DevOps. No solo durante la fase de desarrollo del software, sino también para la gestión de la infraestructura. Se deben automatizar todos los procesos manuales repetitivos y que no aporten valor
- **Trabajar como un solo equipo:** DevOps trata de acabar con los silos y las barreras entre los distintos departamentos. Cada trabajador es responsable de su parte del proyecto.
- **Monitorización y pruebas continuas:** para detectar los errores lo antes posible y mejorar la calidad de los productos generados.

Los beneficios de la utilización de DevOps van orientados a mejorar la calidad del software, reducir el tiempo de entrega y ahorrar esfuerzos/costes en el proceso completo, alineando las estrategias de los equipos de desarrollo y producción/operaciones.

DevOps no se refiere solo a automatización y herramientas, sino que tiene tres componentes fundamentales: **procesos, organización y tecnología**. Si alguno de ellos no está suficientemente alineado con los principios de DevOps, no funcionará y no se obtendrán todos sus beneficios.

Los equipos de trabajo deben ser autónomos, responsables de su trabajo y estar compuestos por miembros de las distintas áreas. También se fomenta la incorporación de los perfiles "T", que son las personas que son buenas en muchas áreas y expertas en al menos una.

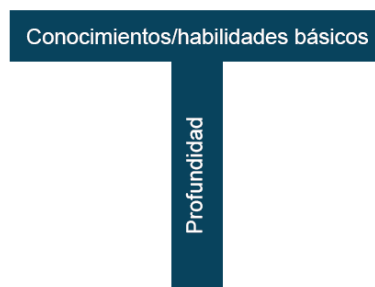


Figura 8. Persona con perfil en forma de T.

Fuente: <https://collegeinfo geek.com/become-t-shaped-person/>

La medición es una de las características clave de una organización DevOps. La organización tiene como objetivo medir tanto como sea posible con el fin de garantizar que los equipos y la organización en su conjunto reciban retroalimentación constante sobre su desempeño.

3.3 Objetivos de DevOps

Los objetivos de DevOps comprenden todo el proceso de entrega. Los principales son los siguientes:

- **Mejora de la frecuencia de despliegue:** en el mundo del desarrollo de software tradicional se suelen compilar muchas versiones, pero se despliegan muy pocas. Uno de los objetivos que se cumplen utilizando metodología DevOps es mejorar la frecuencia y la calidad de los despliegues. Para ello se utilizan herramientas como el Continuous Delivery (entrega continua) y el Continuous Deployment (despliegue continuo), de manera que se compila y prueba el código continuamente cada vez que se realiza un cambio y así, por un lado, se detectan los errores tempranamente y por otro, siempre se tiene lista una versión entregable del software.
- **Mejora del Time to Market:** al utilizar las herramientas anteriores para mejorar la frecuencia de despliegue permite salir al mercado de manera más ágil, detectar los errores antes y empezar a trabajar con productos terminados más rápidamente.
- **Reducción de la tasa de error:** la aplicación de ciclos de monitorización, testeo continuo y corrección de errores detectados, permiten detectar los errores que pasan al entorno de producción y corregirlos produciendo el menor impacto posible y reduciendo así la tasa y los tiempos de error de los productos.
- **Reducción de los tiempos de corrección:** al integrar de manera continua, se trabaja con menos cantidad de cambios en cada delivery y al emplear automatización y monitorización continuas, las correcciones se realizarán en un tiempo más corto al encontrar con mayor facilidad el origen de los fallos.

3.4 El ciclo de vida de DevOps

3.4.1 Ciclo de vida de las metodologías ágiles

Es un ciclo de vida **iterativo e incremental**, con iteraciones cortas (semanas) y sin que dentro de cada iteración tenga que haber fases lineales (tipo cascada), son por tanto muy flexibles. Este tipo de ciclo de vida también se suele llamar **adaptativo u orientado al cambio** ya que responde a niveles altos de cambio y a la participación activa y permanente entre todos los implicados en el proyecto.

Cada tipo de metodología ágil tiene su propio ciclo de vida particularizado, aunque siempre será iterativo e incremental y aumentando la funcionalidad en cada entrega. Existen dos enfoques principales para los distintos tipos de ciclo de vida ágiles:

- Los **enfocados a las iteraciones:** que se basan en ciclos muy rápidos llamados iteraciones, con una duración de entre 1 a 4 semanas como máximo, en las que se debe realizar el trabajo acordado previamente. En la metodología Scrum, por ejemplo, a cada iteración se le denomina Sprint.



Figura 9. Marco de trabajo de Scrum.

Fuente: <https://www.itnove.com/es/agile/coaching-consultoria-scrum-barcelona>

- Los **centrados al flujo del trabajo**: como por ejemplo Kanban, en el que se establecen claramente las limitaciones sobre la secuencia de actividades (concepto de Work In Progress, WIP).

3.4.2 El ciclo de vida DevOps

El ciclo de vida en DevOps se basa en el ciclo de vida de las metodologías ágiles y es por tanto iterativo, compuesto por diferentes fases que buscan siempre el incremento de valor y la mejora continua. Al ser iterativo se pueden realizar múltiples iteraciones en cada fase para mejorar el software y el proceso de entrega continua. Las fases que lo integran forman un bucle infinito donde entran en juego las tareas propias de dos mundos: el de Desarrollo y el de Operaciones. De esta forma, se inicia un flujo de tareas continuas, que comienzan en la fase de análisis y finalizan en la de aprobación del sistema, repitiéndose de nuevo todo el proceso.

El ciclo de vida de DevOps consta de seis fases principales: Planificación, Desarrollo, Pruebas, Despliegue, Operación y Feedback continuo. Cada fase se centra en una etapa específica del ciclo de vida del software y utiliza herramientas y prácticas específicas para acelerar el tiempo de entrega y mejorar la calidad del software:

- **Planificación**: implica la definición de los objetivos del software y la identificación de los requisitos de los usuarios y del negocio. En esta fase, se establecen los requisitos y se planifican las tareas de las siguientes fases, necesarias para desarrollar y desplegar el software.
- **Desarrollo**: implica la escritura del código fuente, sus pruebas unitarias y la creación de la infraestructura necesaria para soportar el software. Los desarrolladores trabajan en equipo utilizando prácticas de control de versiones y colaboración para acelerar el proceso de desarrollo.
- **Pruebas**: esta fase a veces aparece como Integración Continua (CI), Implica la realización de pruebas para verificar que el software cumpla con los requisitos establecidos en la fase de planificación. Las pruebas automatizadas son una parte importante de esta fase y ayudan a identificar los errores y las vulnerabilidades del software antes de que sea desplegado en producción.
- **Despliegue**: implica la entrega del software al entorno de producción. En esta fase, se implementan prácticas de automatización para acelerar el despliegue y mejorar la calidad del software. También se establecen medidas para garantizar la monitorización y el mantenimiento continuo del software en producción.
- **Operación**: una vez que el software está en producción, se monitorea para asegurar que funciona correctamente y se lleva a cabo el seguimiento de problemas, incidentes y cambios, así como proporcionar el mantenimiento necesario.
- **Feedback continuo**: durante todo el ciclo de vida es necesario que exista un flujo continuo de comunicación, por tanto, se recopila información y se obtiene retroalimentación para mejorar el proceso y el software en sí mismo.

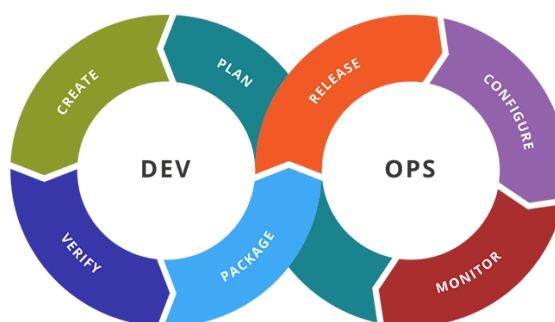


Figura 10. Ciclo de vida de DevOps.

Fuente: <https://www.uxland.es/devops-integracion-continua/>

Lo que pretende conseguir el ciclo de vida DevOps es detectar los errores lo antes posible, utilizando procesos automatizados, para entregar valor al cliente lo antes posible, para ello, en cada fase del ciclo de vida se realizan los tests pertinentes:

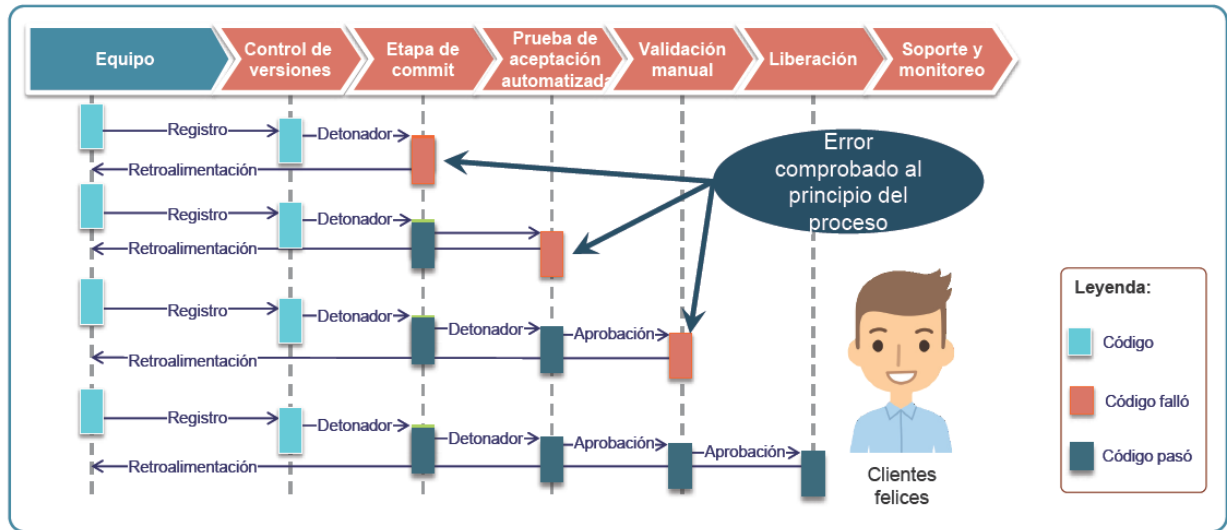


Figura 11. Mejora continua y resolución de problemas
Fuente: <https://www.devopsagileskills.org/>

3.5 Estado del arte de las herramientas DevOps

DevOps no se trata únicamente de herramientas, pero es cierto que son necesarias herramientas especializadas para facilitar y acelerar todos los procesos del flujo DevOps. Actualmente el número de herramientas relacionadas con DevOps ha crecido espectacularmente y cubren cada una de las categorías involucradas en todo el proceso.

En la siguiente imagen se puede apreciar la Tabla Periódica de las herramientas DevOps que genera la empresa Xebialabs y que mantiene actualizada, donde se pueden ver las herramientas más utilizadas actualmente y su funcionalidad:

La 'Tabla Periódica de las herramientas DevOps (v3)' es una grilla de herramientas organizadas por categorías y funcionalidades. Las categorías principales incluyen: Open Source, Free, Freemium, Paid, Enterprise, Source Control Mgmt., Database Automation, Continuous Integration, Testing, Configuration, Deployment, Containers, Release Orchestration, Cloud, AI/Ops, Analytics, Monitoring, Security, Collaboration. Las herramientas listadas incluyen: GitLab, GitHub, Datical, Dt, Sv, Db, Cw, Dp, Jn, Cs, Fn, Ju, Ka, Su, Ch, Tf, Xld, Ud, Ku, Cc, Pr, Al, Os, Ps, At, Rg, Ba, Vs, Se, Jm, Ja, SI, An, Ru, Oc, Go, Ms, Gke, Om, Cp, Cy, It, Nx, Fw, Tr, Tc, Ga, Tn, Tt, Pe, Pu, Pa, Cd, Ec, Ra, Aks, Rk, Sp, Ir, Mg, Bb, Pf, Cr, Cb, Cu, Mc, Lo, Mf, SI, Ce, Eb, Ca, De, Ae, Cf, Hm, Aw, Ls, Xli, Ki, Nr, Dt, Dd, Dd, AppDynamics, El, Ni, Zb, Zn, Cx, Sg, Bd, Sr, Hv, Sw, Jr, Tl, Sl, St, Cn, Ry, Ac, Og, Pd, Sn, Tw, Ck, Vc, Ff.

Figura 12. Tabla periódica de las herramientas DevOps.
Fuente: <https://xebialabs.com/periodic-table-of-devops-tools/>

En los siguientes capítulos se muestran algunas de las herramientas más importantes utilizadas actualmente en cada una de las categorías involucradas en los procesos DevOps.

3.5.1 Gestión de la configuración del código fuente

Los sistemas de control de código fuente o SCM (*Source Code Management*) se utilizan para llevar el control sobre los cambios realizados en el código fuente por los distintos usuarios: modificaciones, creación y merge de ramas, resolución de conflictos entre versiones de ficheros, etc.

Algunos de los SCM más utilizados son los siguientes:

Nombre de la herramienta	Descripción	Web de referencia
Bitbucket	Herramienta de control de versiones distribuida para equipos pequeños y medianos.	https://bitbucket.org/
CVS	Sistema de control de versiones que permite la colaboración entre varios desarrolladores en un mismo proyecto.	https://www.nongnu.org/cvs/
Git	Sistema de control de versiones distribuido que permite el control de versiones y la gestión de código fuente.	https://git-scm.com/
GitHub	Plataforma de Git para alojamiento de código, seguimiento de problemas, integración con herramientas de CI/CD y análisis de seguridad.	https://github.com/
GitLab	Plataforma de Git que incluye características como gestión de repositorios de código, seguimiento de problemas, CI/CD integrado y análisis de seguridad.	https://about.gitlab.com/
Mercurial	Sistema de control de versiones distribuido que permite el control de versiones y la gestión de código fuente. Admite tanto Git como Mercurial.	https://www.mercurial-scm.org/
SourceSafe	Sistema de control de versiones centralizado que permite el control de versiones y la gestión de configuración.	https://docs.microsoft.com/en-us/azure/devops/server/source-safe/vss-2019?view=azure-devops
Subversion	Sistema de control de versiones centralizado que utiliza un modelo cliente-servidor para administrar versiones de archivos y directorios.	https://subversion.apache.org/
Team Foundation Server	Plataforma de Microsoft que proporciona gestión de repositorios de código, seguimiento de problemas, integración con herramientas de CI/CD y análisis de seguridad.	https://visualstudio.microsoft.com/tfs/
TortoiseSVN	Cliente de Subversion para Windows que permite la gestión de versiones de archivos y carpetas.	https://tortoisesvn.net/

Tabla 2. Herramientas Open Source para la gestión de la configuración del código fuente (SCM)

3.5.2 Integración Continua (CI)

Es una práctica de desarrollo que se basa en la integración frecuente del código fuente en un mismo repositorio compartido. Cada cambio introducido es verificado por una construcción automatizada, permitiendo a los equipos detectar los problemas más rápidamente.

Algunas de las herramientas que facilitan la CI más utilizadas son las siguientes:

Nombre de la herramienta	Descripción	Web de referencia
Bamboo	Herramienta de integración continua para proyectos de gran escala. Proporciona un flujo de trabajo de construcción y distribución completamente automatizado.	https://www.atlassian.com/software/bamboo
CircleCI	Plataforma de integración continua en la nube que admite varios lenguajes de programación y marcos de prueba. Permite una integración rápida y una implementación continua.	https://circleci.com/
GitLab CI	Parte de la plataforma GitLab, GitLab CI es una herramienta de integración continua y entrega continua. Es muy flexible y se puede utilizar tanto para proyectos pequeños como grandes.	https://docs.gitlab.com/ee/ci/
Jenkins	Herramienta de integración continua de código abierto muy popular. Es altamente personalizable y admite una gran cantidad de complementos.	https://www.jenkins.io/
Travis CI	Plataforma de integración continua basada en la nube. Es fácil de configurar y utilizar, y es compatible con varios lenguajes de programación y sistemas operativos.	https://www.travis-ci.com/
Buddy	Herramienta de integración continua y entrega continua diseñada para equipos de desarrollo web y móvil. Es fácil de usar y se integra con varias plataformas en la nube.	https://buddy.works/
Codship	Herramienta de integración continua basada en la nube que proporciona una integración y entrega continua rápidas y fáciles. Admite varios lenguajes de programación y sistemas operativos.	https://www.cloudbees.com/products/codship
TeamCity	Herramienta de integración continua y entrega continua de JetBrains. Es altamente personalizable y admite una amplia gama de tecnologías.	https://www.jetbrains.com/teamcity/
Travis CI	Plataforma de integración continua basada en la nube. Es fácil de configurar y utilizar, y es compatible con varios lenguajes de programación y sistemas operativos.	https://www.travis-ci.com/
Buddy	Herramienta de integración continua y entrega continua diseñada para equipos de desarrollo web y móvil. Es fácil de usar y se integra con varias plataformas en la nube.	https://buddy.works/

Tabla 3. Herramientas Open Source para la Integración Continua (CI)

3.5.3 Testing

Las herramientas de testing son imprescindibles para validar el código antes y después de integrarlo. Uno de los pilares de DevOps es la ejecución automatizada de los tests para detectar errores lo antes posible.

Existen muchos tipos de pruebas que se pueden acometer durante el ciclo de vida completo de DevOps. A continuación se detallan las más importantes:

- **Pruebas unitarias:** son pruebas que se centran en comprobar el correcto funcionamiento de una unidad de código fuente, es decir, un pequeño fragmento de código. Algunas de las herramientas Open Source más utilizadas para este tipo de pruebas son: JUnit para Java, NUnit para .NET, pytest para Python.

- **Pruebas de integración:** son pruebas que se enfocan en comprobar el correcto funcionamiento de varias unidades de código juntas, asegurándose de que se integren correctamente. Algunas de las herramientas Open Source más utilizadas para este tipo de pruebas son: Selenium para pruebas de integración de aplicaciones web, Cucumber para pruebas de integración de software, Postman para pruebas de integración de APIs.
- **Pruebas de carga y rendimiento:** son pruebas que se realizan para evaluar el comportamiento de una aplicación en situaciones de alta carga o tráfico. Algunas de las herramientas Open Source más utilizadas para este tipo de pruebas son: Apache JMeter para pruebas de carga y rendimiento, Gatling para pruebas de rendimiento, Locust para pruebas de carga.
- **Pruebas de usabilidad:** son pruebas manuales o automatizadas que se utilizan para evaluar la facilidad de uso y accesibilidad de la aplicación por parte del usuario. Algunas de las herramientas Open Source más utilizadas para este tipo de pruebas son: Open Web Analytics para análisis de sitios web, Selenium para la automatización de pruebas de software y Appium que es como Selenium para aplicaciones móviles.
- **Pruebas de seguridad:** son pruebas que se centran en identificar vulnerabilidades y posibles brechas de seguridad en una aplicación o sistema. Se profundizará en el tema en los siguientes capítulos de esta memoria de TFG.
- **Pruebas de aceptación:** son pruebas que se realizan para verificar que una aplicación cumple con los requisitos del usuario y funciona correctamente. Algunas de las herramientas Open Source más utilizadas para este tipo de pruebas son: Robot Framework para pruebas de aceptación, FitNesse para pruebas de aceptación de software, Gauge para pruebas de aceptación de aplicaciones web.

3.5.4 Entrega Continua y Despliegue Continuo

La entrega continua (Continuous Delivery, CD) y el despliegue continuo (Continuous Deployment, CD) tienen como objetivo hacer llegar al cliente final lo antes posible y con calidad, las modificaciones que se han implementado en desarrollo.

La entrega continua es el proceso automatizado de construcción, pruebas y preparación del software para ser liberado en cualquier momento. La idea es tener un software listo para ser entregado a producción en todo momento. Para ello, se utilizan herramientas que permiten la integración y automatización de los procesos, lo que facilita la realización de pruebas y asegura que el software esté listo para ser liberado.

Por otro lado, el despliegue continuo es la práctica de automatizar todo el proceso de liberación y despliegue del software en producción, eliminando la intervención manual. Con esta práctica, cualquier cambio que supere las pruebas de integración y funcionales, puede ser liberado y desplegado de forma inmediata.

Por tanto, la diferencia más notable entre ambos es la forma que se hace el despliegue, automática o manual. Mientras que en el despliegue continuo cada versión se despliega de forma automática, en la entrega continua, este procedimiento tiene un paso manual que es la aprobación del cambio.

Las herramientas Open Source más empleadas para la entrega continua y el despliegue continuo son:

Nombre de la herramienta	Descripción	Web de referencia
Ansible	Plataforma de automatización que permite la entrega continua y el despliegue continuo	https://www.ansible.com/
Bamboo	Herramienta de integración y entrega continua que automatiza la compilación, prueba y publicación de aplicaciones	https://www.atlassian.com/software/bamboo
CircleCI	Herramienta de integración continua y entrega continua en la nube	https://circleci.com/
GitLab CI/CD	Plataforma de entrega continua que se integra con GitLab, que incluye pruebas y despliegue automático	https://about.gitlab.com/stages-devops-lifecycle/continuous-integration/
Jenkins	Herramienta de integración continua y entrega continua que automatiza la construcción, prueba y despliegue de aplicaciones	https://www.jenkins.io/
Octopus Deploy	Herramienta de despliegue automatizado y entrega continua para aplicaciones .NET	https://octopus.com/
Spinnaker	Plataforma de entrega continua y orquestación de despliegue que se integra con varias herramientas de infraestructura en la nube	https://www.spinnaker.io/
Tekton	Kit de herramientas para construir y ejecutar pipelines de entrega continua y despliegue continuo	https://tekton.dev/
Travis CI	Herramienta de integración continua y entrega continua en la nube para proyectos alojados en GitHub	https://travis-ci.com/
Wercker	Herramienta de entrega continua y despliegue continuo que se integra con varias plataformas en la nube	https://app.wercker.com

Tabla 4. Herramientas Open Source para la Entrega y el Despliegue Continuo (CD)

3.5.5 Monitorización

La monitorización es esencial para saber el estado de los sistemas y servicios en todo momento. Es necesario recopilar esa información, mostrarla de una manera adecuada y que todos los usuarios tengan acceso a ella.

La monitorización es el proceso de recolectar y analizar datos sobre los sistemas y aplicaciones en tiempo real con el fin de detectar y solucionar problemas de manera proactiva, y también para tomar decisiones informadas sobre la optimización del rendimiento.

La monitorización puede abarcar diferentes aspectos del ciclo de vida de una aplicación, incluyendo la infraestructura, los servicios web, la base de datos y las métricas de rendimiento de la aplicación. En DevOps se intenta automatizar al máximo la recopilación y el análisis de datos, para generar alertas automáticas en caso de que se detecten problemas o se excedan ciertos umbrales de rendimiento.

Algunas de las herramientas más utilizadas en la monitorización son las siguientes:

Nombre de la herramienta	Descripción	Web de referencia
Grafana	Herramienta de visualización y análisis de datos que permite monitorizar métricas de diferentes fuentes.	https://grafana.com/

Nombre de la herramienta	Descripción	Web de referencia
Icinga	Plataforma de monitorización de infraestructuras de tecnología de la información que proporciona información en tiempo real sobre el estado de los servicios y aplicaciones.	https://icinga.com/
Nagios	Herramienta de monitorización de sistemas que comprueba la disponibilidad de los servicios de red y notifica al equipo en caso de problemas.	https://www.nagios.org/
Netdata	Sistema de monitorización en tiempo real para sistemas Linux	https://www.netdata.cloud/
Prometheus	Sistema de monitorización y alerta de métricas y series temporales que permite recopilar datos de múltiples fuentes.	https://prometheus.io/
Sensu	Plataforma de monitorización de infraestructuras de TI que ofrece un conjunto de herramientas para supervisar y recopilar datos de diferentes sistemas y dispositivos.	https://sensu.io/
Shinken	Sistema de monitorización de redes y aplicaciones compatible con Nagios	https://www.shinken-monitoring.org/
TICK Stack	Conjunto de herramientas para monitorización y análisis de series temporales	https://www.influxdata.com/time-series-platform/
Zabbix	Herramienta de monitorización de redes y sistemas que recopila información sobre el estado de los dispositivos y notifica al equipo en caso de problemas.	https://www.zabbix.com/
Zenoss	Plataforma de monitorización de infraestructura de TI	https://www.zenoss.com/

Tabla 5. Herramientas Open Source para la Monitorización

3.5.6 Alertas

Monitorizar no es suficiente, es necesario contar con un servicio de alertas que informe de los problemas a cualquier hora del día y todos los días del año.

La generación de alertas se basa en la definición de métricas y umbrales de rendimiento que indican cuándo se debe generar una alerta. Por ejemplo, se puede establecer un umbral de uso de CPU para que, cuando se supere cierto valor, se genere una alerta para investigar y solucionar el problema.

Las herramientas de monitoreo y supervisión de la infraestructura y el software son clave en la generación de alertas. Estas herramientas son capaces de recopilar datos en tiempo real y emitir alertas cuando se detectan problemas o se exceden los umbrales de rendimiento establecidos.

Algunas de las herramientas más utilizadas para la generación de alertas son las siguientes:

Nombre de la herramienta	Descripción	Web de referencia
Alertmanager	Herramienta de código abierto para la gestión de alertas que se integra con sistemas de monitorización como Prometheus y Graphite.	https://prometheus.io/docs/alerting/alertmanager/
Cabot	Plataforma de monitorización y alerta de código abierto diseñada para supervisar servicios y enviar alertas cuando algo sale mal.	https://cabotapp.com/

Nombre de la herramienta	Descripción	Web de referencia
Graylog	Plataforma de gestión de registros y análisis de logs que permite la generación de alertas en tiempo real.	https://www.graylog.org/
Icinga	Herramienta de monitorización de sistemas de código abierto que permite la generación de alertas en función de diferentes parámetros.	https://icinga.com/
Nagios	Sistema de monitorización de redes de código abierto que permite la generación de alertas en función de los estados de los servicios y los hosts.	https://www.nagios.org/
Netdata	Herramienta de monitorización y análisis de rendimiento de sistemas de código abierto que ofrece alertas en tiempo real.	https://www.netdata.cloud/
Prometheus	Sistema de monitorización y alerta de código abierto diseñado para recopilar métricas de sistemas y servicios.	https://prometheus.io/
Sensu	Plataforma de monitorización de código abierto que permite la generación de alertas en tiempo real y la automatización de tareas.	https://sensu.io/
Shinken	Sistema de monitorización de código abierto que permite la generación de alertas en función de los estados de los servicios y los hosts.	https://shinken.io/
Zabbix	Plataforma de monitorización de redes y sistemas de código abierto que permite la generación de alertas en función de diferentes parámetros.	https://www.zabbix.com/

Tabla 6. Herramientas Open Source para la generación de Alertas

3.5.7 Seguridad

La seguridad en DevOps es fundamental, sin embargo, al buscar desarrollos ágiles se puede correr el riesgo de dar menos importancia a la seguridad. DevSecOps se ocupa de incluir la seguridad desde el principio de todo el proceso. Los detalles de DevSecOps se mostrarán en los siguientes capítulos de la memoria.

3.5.8 Virtualización y containerización

La virtualización es una tecnología que permite que un equipo anfitrión (host) con un sistema operativo propio pueda ejecutar una o más máquinas virtuales con sistemas operativos clientes. Para ello un programa crea un entorno virtual (máquina virtual) donde se sintetiza un entorno de computación real (NIC virtual, BIOS, tarjeta de sonido, vídeo, etc.). Esto es posible gracias a una capa de software que gestiona estos recursos reales en el equipo anfitrión y los reparte dinámicamente entre las distintas máquinas virtuales hospedadas en dicho equipo. A este software intermedio se le denomina hypervisor.

Los contenedores son también una tecnología de virtualización, pero no necesitan un programa hypervisor ni máquinas virtuales para ser ejecutados, sino que se ejecutan directamente sobre el kernel del sistema operativo anfitrión (OS-level virtualization), que se encarga de ejecutar los distintos contenedores y de aislarlos unos de otros.

VMs vs Docker

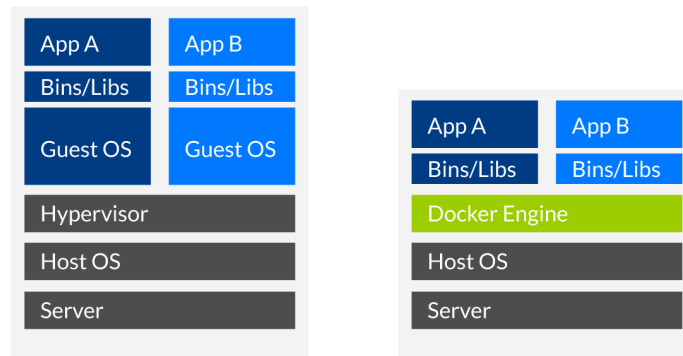


Figura 13. Máquinas virtuales Vs Contenedores

Fuente: <https://www.lomasnuevo.net/cloud/maquinas-virtuales-vs-contenedores/>

El uso de contenedores proporciona una serie de ventajas con respecto al uso de máquinas virtuales:

- Menor tamaño: al no contener al sistema operativo son ficheros mucho más pequeños.
- Menor uso de memoria: al no tener que ejecutar el sistema operativo necesita menos memoria. Por esta razón (y el punto anterior), se pueden ejecutar de 10 a 100 veces más contenedores que máquinas virtuales en el mismo equipo.
- Provisión de recursos: al crear una máquina virtual se debe provisionar el espacio que ocupará en el disco duro, la memoria reservada, el número de CPUs, etc. Este paso no es necesario en los contenedores.
- Arranque/parada más rápido: los contenedores pueden iniciarse en segundos mientras que las máquinas virtuales deben arrancar el sistema operativo y pueden tardar minutos.

También alguna desventaja como:

- No pueden ejecutar un programa de un sistema operativo distinto al de la máquina host, que con máquinas virtuales sí se puede.
- Peor aislamiento de ataques: las máquinas virtuales proporcionan abstracción a nivel de hardware, de manera que un ataque puede afectar únicamente a la máquina virtual comprometida, aislando a todas las demás que se están ejecutando en el mismo host.

Algunas de las herramientas más utilizadas para máquinas virtuales las siguientes:

Nombre de la herramienta	Descripción	Web de referencia
KVM	Plataforma de virtualización de código abierto que permite la ejecución de múltiples máquinas virtuales en un host.	https://www.linux-kvm.org/page/Main_Page
OpenStack	Plataforma de nube privada de código abierto que permite el despliegue y la gestión de máquinas virtuales.	https://www.openstack.org/
QEMU	Herramienta de virtualización de sistema completa que puede ejecutar una variedad de sistemas operativos invitados.	https://www.qemu.org/
VirtualBox	Plataforma de virtualización de código abierto que permite la ejecución de múltiples sistemas operativos en una sola máquina física.	https://www.virtualbox.org/

Nombre de la herramienta	Descripción	Web de referencia
Virt-manager	Aplicación de gestión de máquinas virtuales basada en libvirt que permite la gestión de máquinas virtuales en hosts locales y remotos.	https://virt-manager.org/
VMware ESXi	Plataforma de virtualización de servidor de tipo hipervisor que permite la ejecución de múltiples sistemas operativos en un único host.	https://www.vmware.com/products/esxi-and-esx.html
Xen	Hipervisor de código abierto que permite la ejecución de múltiples sistemas operativos invitados en una sola máquina física.	https://xenproject.org/

Tabla 7. Herramientas Open Source para la Virtualización

Para la creación de contenedores las herramientas más utilizadas son:

Nombre de la herramienta	Descripción	Web de referencia
Docker	Plataforma para desarrollar, enviar y ejecutar aplicaciones en contenedores.	https://www.docker.com/
LXD	Sistema de contenedores ligeros que se ejecuta directamente en el sistema operativo anfitrión y que ofrece una experiencia de máquina virtual sin el sobrecoste típico de la virtualización tradicional.	https://linuxcontainers.org/lxd/
Proxmox VE	Plataforma de virtualización de servidor de código abierto que integra KVM y contenedores LXC.	https://www.proxmox.com/en/proxmox-ve

Tabla 8. Herramientas Open Source para la Containerización

3.5.9 Orquestación

Actualmente las aplicaciones suelen ser complejas y por regla general no basta con desplegar un solo contenedor en producción, lo habitual es tener que desplegar varios, por ejemplo, un contenedor para el Front-End, otro para la base de datos, otro para determinados servicios, etc.

Para solucionar esta necesidad surgió la orquestación de contenedores, esto es, el proceso de gestionar y coordinar la implementación, el escalado y la gestión de contenedores en un entorno de aplicaciones distribuidas. La orquestación de contenedores se utiliza para automatizar la implementación de contenedores, facilitar la escalabilidad y la administración, y mejorar la disponibilidad y la resiliencia de las aplicaciones.

Algunas de las herramientas más utilizadas para la orquestación son las siguientes:

Nombre de la herramienta	Descripción	Web de referencia
Docker Swarm	Herramienta de orquestación de contenedores integrada en Docker Engine que permite gestionar y escalar aplicaciones en múltiples nodos.	https://docs.docker.com/engine/swarm/
Kubernetes	Plataforma de orquestación de contenedores de código abierto que permite gestionar y escalar aplicaciones en múltiples nodos.	https://kubernetes.io/

Nombre de la herramienta	Descripción	Web de referencia
Mesos	Sistema de gestión de recursos de código abierto que permite gestionar y orquestar aplicaciones distribuidas en múltiples nodos.	http://mesos.apache.org/
Nomad	Sistema de orquestación de contenedores y aplicaciones que permite gestionar cargas de trabajo en múltiples entornos, desde máquinas virtuales hasta nubes públicas.	https://www.nomadproject.io/
Rancher	Plataforma de gestión de contenedores de código abierto que incluye una herramienta de orquestación para Kubernetes y Docker Swarm.	https://rancher.com/
Swarmkit	Biblioteca de orquestación de contenedores de código abierto integrada en Docker Engine que permite gestionar y escalar aplicaciones en múltiples nodos.	https://docs.docker.com/engine/swarm/swarmkit/
Apache Aurora	Sistema de orquestación de contenedores de código abierto que permite gestionar y programar aplicaciones en múltiples nodos.	https://aurora.apache.org/
AWS ECS	Servicio de orquestación de contenedores en la nube de Amazon que permite gestionar y escalar aplicaciones en múltiples nodos.	https://aws.amazon.com/ecs/
DC/OS	Sistema operativo de código abierto que incluye una herramienta de orquestación de contenedores que permite gestionar y escalar aplicaciones en múltiples nodos.	https://dcos.io/
OpenShift	Plataforma de contenedores de código abierto que incluye una herramienta de orquestación de contenedores basada en Kubernetes.	https://www.openshift.com/

Tabla 9. Herramientas Open Source para la Orquestación

3.5.10 Comunicación y colaboración

No son estrictamente herramientas de DevOps, sino más bien de gestión de proyectos y de trabajo en equipo, sin embargo, son imprescindibles para que la información fluya y se comparta por todos los miembros de la organización involucrados en los procesos DevOps. La comunicación efectiva y la colaboración son esenciales para el éxito de DevOps, ya que permiten que los equipos trabajen juntos de manera más eficiente, reduzcan los silos de información y tomen decisiones más informadas y precisas.

La comunicación en DevOps puede tomar muchas formas, desde reuniones regulares para la planificación y la revisión de código, hasta herramientas de mensajería instantánea y foros de discusión en línea. También es común el uso de herramientas de seguimiento de problemas y proyectos que permiten a los equipos colaborar en el seguimiento y la solución de problemas.

Por otro lado, la colaboración en DevOps implica trabajar juntos en un entorno de equipo y compartir responsabilidades para lograr un objetivo común. Los equipos de desarrollo y operaciones trabajan juntos para asegurarse de que los cambios en el código se implementen y se entreguen de manera segura y confiable. La colaboración también implica compartir conocimientos y habilidades para que todos los miembros del equipo estén capacitados para tomar decisiones y resolver problemas.

Algunas de las herramientas Open Source más utilizadas para mejorar la comunicación y fomentar la colaboración en DevOps son las siguientes:

Nombre de la herramienta	Descripción	Web de referencia
Gitter	Plataforma de mensajería instantánea para equipos de desarrollo	https://gitter.im/
GitLab	Plataforma de colaboración y gestión de proyectos basada en Git	https://about.gitlab.com/
Mattermost	Plataforma de mensajería de equipo y colaboración en tiempo real	https://mattermost.com/
Microsoft Teams	Plataforma de colaboración y comunicación en tiempo real de Microsoft	https://www.microsoft.com/en-us/microsoft-teams/group-chat-software
Rocket.Chat	Plataforma de chat, voz y videoconferencia para equipos de trabajo	https://rocket.chat/
Slack	Plataforma de mensajería y colaboración en tiempo real para equipos de trabajo	https://slack.com/
Trello	Plataforma de gestión de proyectos basada en tableros Kanban	https://trello.com/
Zulip	Plataforma de chat y colaboración en tiempo real con organización de mensajes por temas	https://zulip.com/
Zoom	Plataforma de videoconferencia y reuniones virtuales en tiempo real	https://zoom.us/
Jitsi Meet	Plataforma de videoconferencia y reuniones virtuales de código abierto	https://jitsi.org/jitsi-meet/

Tabla 10. Herramientas Open Source para la Comunicación y la Colaboración

4 Introducción a DevSecOps

4.1 ¿Qué es DevSecOps?

DevSecOps es una metodología de desarrollo de software que busca integrar la seguridad en todas las etapas del proceso de desarrollo y entrega de software. En lugar de considerar la seguridad como una etapa posterior, que se realiza al final del desarrollo del software, DevSecOps busca abordar los problemas de seguridad desde el inicio del ciclo de vida del desarrollo del software, integrando la seguridad en todas las fases del proceso de desarrollo de software y fomentando la colaboración entre equipos para mejorar la seguridad y la calidad del software entregado.

El término DevSecOps se compone de tres palabras: Dev, que se refiere al equipo de desarrollo; Sec, que representa la seguridad; y Ops, que se refiere al equipo de operaciones. La idea detrás de DevSecOps es que los equipos de desarrollo, seguridad y operaciones trabajen juntos desde el inicio del proceso de desarrollo de software, compartiendo responsabilidades y colaborando en la integración de la seguridad en todas las fases del ciclo de vida del desarrollo de software.

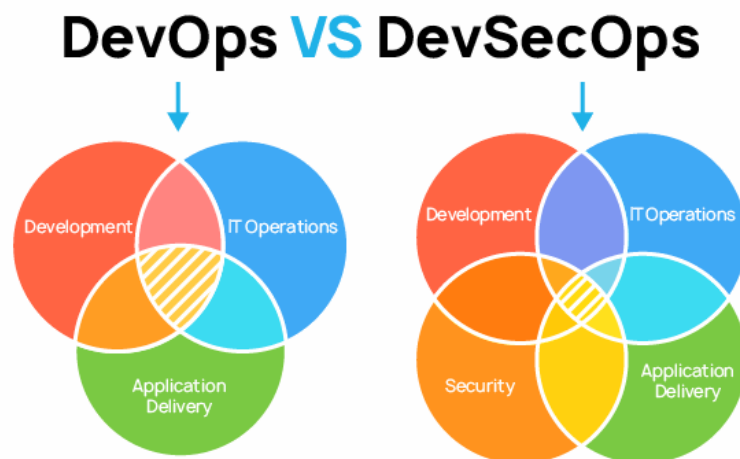


Figura 14: DevOps vs DevSecOps

Fuente: <https://blog.educacionit.com/2021/06/27/serie-de-devops-101-devsecops/>

Antes de la implementación de la cultura DevSecOps, normalmente se aplicaban los controles de seguridad en la fase de despliegue del software y desafortunadamente, en ese momento ya suele ser demasiado tarde. El equipo encargado de la seguridad puede encontrar problemas en esa fase reportándolos a desarrollo, que tendrá que aplicar los parches y correcciones necesarios, que deberán ser probados de nuevo, incurriendo en más retrasos en los tiempos de entrega y el sobrecoste correspondiente.

Para mitigar este problema ha surgido DevSecOps, que fomenta introducir la seguridad desde las primeras fases del ciclo de desarrollo, minimizando la cantidad de vulnerabilidades desde el principio, para poder enviar a producción un producto seguro, de una manera automatizada y en el menor tiempo posible.

Cada año, el Synopsys Cybersecurity Research Centre (CyRC) genera un informe, el “Open Source Security and Risk Analysis (OSSRA)”, sobre el estado de la seguridad del código fuente Open Source y las empresas que lo utilizan. En el informe de este año, del 22 de febrero de 2023, se aprecia que la cantidad de aplicaciones con vulnerabilidades de alto riesgo cayó a su nivel más bajo en cuatro años, demostrando un creciente interés por la seguridad de las aplicaciones.

Este estudio anual, basado en auditorías de más de 1700 aplicaciones, encontró que:

- Casi todos los programas de software (96%) incluían algún tipo de componente de software de código abierto
- Con una base de código promedio que consiste en un 76% de código fuente abierto.
- La cantidad de bases de código con al menos una vulnerabilidad se mantuvo mayormente estable durante los últimos tres años, en poco más del 80 % (84 % en 2022)
- La cantidad de aplicaciones con vulnerabilidades de alto riesgo se redujo a aproximadamente la mitad (48 %) de todas aplicaciones probadas, desde un pico de alrededor del 60% en 2020.

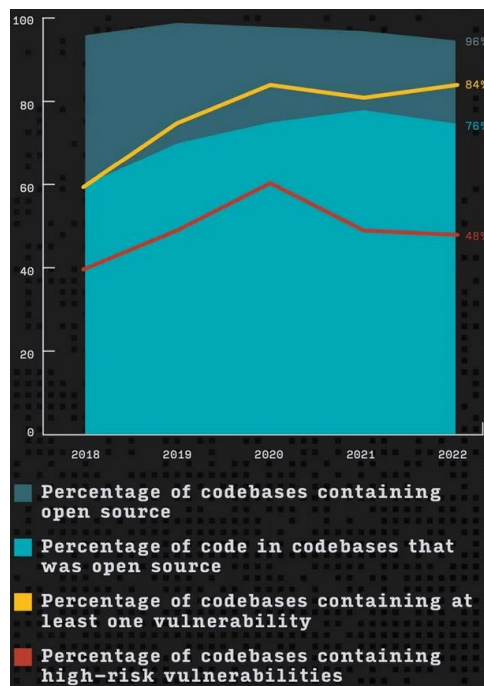


Figura 15. Open Source codebase vulnerabilities

Fuente: Open Source Security and Risk Analysis (OSSRA) 2023

A diferencia de DevOps, DevSecOps no tiene un manifiesto de referencia, hay varios, pero ninguno está estandarizado. Uno de los más referenciados es el llamado "The DevSecOps Manifiesto" [30]. Este manifiesto fue creado en 2018 por un grupo de expertos en seguridad y desarrollo de software para promover el enfoque de seguridad en el ciclo de vida de desarrollo de software (SDLC) y consta de cuatro principios fundamentales:

- **Los equipos de seguridad y de desarrollo de software trabajan juntos en todo momento.** Esto implica que la seguridad debe ser una consideración clave en todas las etapas del SDLC y que los equipos de seguridad deben trabajar en estrecha colaboración con los equipos de desarrollo de software.
- **La seguridad es responsabilidad de todos.** Todos los miembros del equipo, no solo los especialistas en seguridad, deben ser responsables de garantizar la seguridad del software.
- **Automatización de la seguridad.** La automatización de la seguridad puede ayudar a identificar y remediar las vulnerabilidades de seguridad de forma más rápida y eficiente.
- **Medición, aprendizaje y mejora continua.** Es importante medir el éxito de las iniciativas de seguridad y aprender de los errores para mejorar continuamente.

El manifiesto DevSecOps es una declaración de intenciones que destaca la importancia de la seguridad en el desarrollo de software y aboga por la colaboración, la automatización y la mejora continua.

4.2 DevSecOps vs SecDevOps

A primera vista parece que ambos términos quieren decir lo mismo, sin embargo, aunque su finalidad es muy parecida, incrementar la seguridad del software que se va a desarrollar, lo implementan de diferentes maneras:

- **DevSecOps:** como se explicó en el apartado anterior, consiste en la integración de procesos de seguridad, de auditorías de seguridad activa y de pruebas de seguridad en el flujo de trabajo de desarrollo ágil de DevOps. En lugar de aplicar la seguridad al producto final, tiene como objetivo tener la seguridad integrada en el producto aplicando el concepto de seguridad por diseño, lo que significa que la seguridad se toma en consideración desde el principio y durante todo el ciclo de vida del software que estemos desarrollando. Resumiendo, consiste en agregar una etapa extra de verificaciones de seguridad en cada uno de los pasos del ciclo DevOps (ver Figura 16. Pruebas en cada fase del SSDLC.).
- **SecDevOps (Security DevOps):** su objetivo principal es asegurar que los aspectos de seguridad se aborden desde las primeras etapas del desarrollo, en lugar de ser considerados como un paso posterior, involucrando a los equipos de seguridad de manera proactiva. Esto significa que la velocidad de lanzamiento ya no es una prioridad, lo que puede tener un efecto negativo en el proceso de desarrollo, debido a que se centra demasiado en la seguridad y no en que el producto sea entregado.

4.3 Ciclo de vida de desarrollo seguro (SSDLC)

El SSDLC (o S-SDLC) es un enfoque de desarrollo de software que tiene en cuenta la seguridad desde el principio (es el concepto de “Security By Design”), integrando prácticas de seguridad en todo el ciclo de vida del desarrollo de software. Su adopción permite a los equipos de desarrollo identificar y mitigar los riesgos de seguridad desde el inicio del proceso de desarrollo para garantizar que el software sea seguro y confiable, y minimizar el riesgo de vulnerabilidades y brechas de seguridad.

El SSDLC consta de una serie de fases que se ejecutan en secuencia, y cada fase tiene una serie de actividades y tareas específicas. Las fases del SSDLC incluyen la planificación, el diseño, la implementación, las pruebas, el despliegue y el mantenimiento. Cada fase se centra en diferentes aspectos de la seguridad del software, desde la identificación de riesgos hasta la implementación de medidas de seguridad y la realización de pruebas para garantizar que el software sea seguro y confiable.

Las diferentes fases son las siguientes:

- **Planificación:** implica identificar los objetivos de seguridad y las restricciones del proyecto, así como definir los requisitos de seguridad para el software.
- **Diseño:** implica la creación de un diseño seguro que cumpla con los requisitos de seguridad definidos en la fase de planificación.
- **Desarrollo:** implica la escritura y la integración de código seguro.
- **Pruebas:** implica la realización de pruebas para verificar que el software cumpla con los requisitos de seguridad.
- **Despliegue:** implica la implementación del software en un entorno de producción seguro.
- **Operación:** implica la ejecución del sistema en el entorno de producción.
- **Mantenimiento:** implica la monitorización y el mantenimiento continuo del software para asegurarse de que siga siendo seguro y confiable a lo largo del tiempo.

En cada una de las etapas del SSDLC se debe tener en cuenta la seguridad antes de pasar a la siguiente etapa. Las actividades principales relacionadas con la seguridad que se deben acometer en cada etapa son las siguientes:

- **Planificación y diseño:** en esta fase es donde se analiza el modelo de amenazas, también conocido como Threat Modeling. Se valoran las amenazas que se pueden producir en funcionalidades críticas de la aplicación como la autenticación de usuarios, servidores externos,

cifrado, etc. Aquí se generarán “tickets” también conocidos como “stories” destinados específicamente a la seguridad.

- **Desarrollo:** en esta etapa, el equipo de seguridad se convierte en formador y proveedor explicando a los integrantes del equipo de desarrollo cómo utilizar herramientas de seguridad, como las utilizadas para el análisis de código estático, pero sobre todo a familiarizar a los desarrolladores en términos relacionados con la seguridad como por ejemplo qué son los ataques SQLi, XSS, DDoS, etc.
- **Pruebas:** en esta fase se ejecutan los tests unitarios, integración y QA. Sin embargo, en DevSecOps también se realizan tests que verifiquen aspectos de seguridad. Estas pruebas se realizan de forma automática. Las herramientas para esta fase son variadas y dependen del entorno y lenguaje de programación utilizado para el desarrollo.
 - **Package:** esta etapa representa la unificación de todo el código en un sólo paquete o artefacto, ya sea un programa compilado, imagen o paquete comprimido. Un escaneo de seguridad para las librerías externas donde se detecten posibles vulnerabilidades, sería la operativa de seguridad a aplicar en este caso y si se utilizan tecnologías como Docker, se hace necesario evaluar la seguridad de las imágenes.
 - **Release:** en esta fase es donde se ubica el artefacto generado en la fase de “package” en algún repositorio central.
- **Despliegue:** la aplicación se despliega o instala en un entorno preparado para realizar pruebas funcionales. Podrán existir varios entornos de pruebas específicos para que cada equipo pueda trabajar en una parte concreta de la aplicación. Esta fase se conoce también como “pase”, que puede ser a un entorno de “preproducción” o “producción”. Aquí es donde se integra el equipo de QA para realizar los test de calidad y es donde se realiza un análisis dinámico de la aplicación por medio de herramientas DAST.
- **Operación:** en esta fase la aplicación ya está desplegada y funcionando. Ahora se realizan pruebas RedTeam de seguridad, resiliencia (Chaos Engineering, Circuit Breaker, etc.), etc.
- **Feedback continuo:** en esta fase se analizan los logs de forma automática para detectar posibles ataques, así como el estado de uso de la infraestructura de hardware (uso de CPU, memoria, disco duro, etc.) por si se estuviera produciendo un ataque en ese instante.

En la siguiente imagen se pueden ver algunas de las pruebas principales que se realizan en cada fase del SSDLC:

DevSecOps

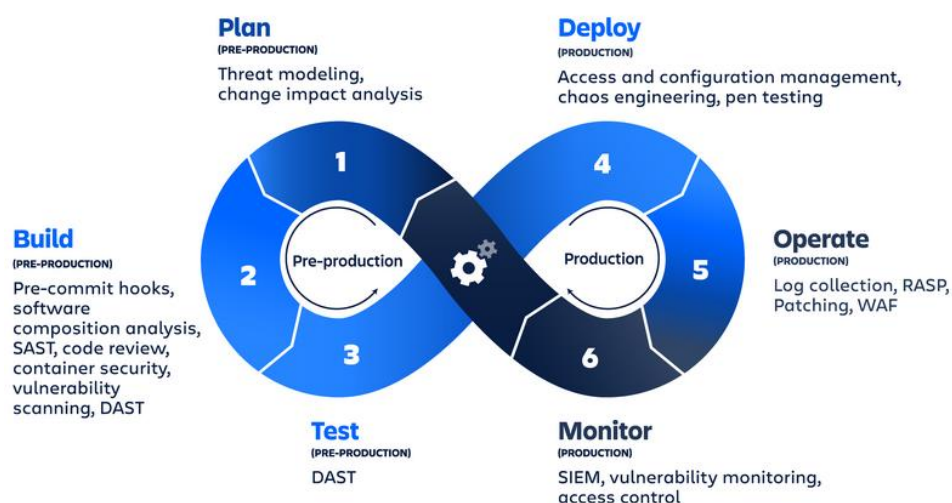


Figura 16. Pruebas en cada fase del SSDLC.

Fuente: <https://www.atlassian.com/es/devops/devops-tools/devsecops-tools>

4.4 Herramientas de pruebas de seguridad en software

La pirámide AST (Application Security Testing) es una estrategia de pruebas de seguridad de aplicaciones que se utiliza en DevSecOps para garantizar que la aplicación se desarrolla con seguridad desde el inicio del proceso de desarrollo y que se realizan pruebas de seguridad de manera constante durante todo el ciclo de vida de la aplicación.

Esta pirámide se compone de las siguientes tres capas principales:

- **Pruebas de seguridad de código fuente (Source Code):** en esta capa situada en la base de la pirámide, se realizan pruebas de seguridad de código fuente mediante el uso de herramientas de análisis estático de código (Static Application Security Testing, SAST) y análisis dinámico de código (Dynamic Application Security Testing, DAST) para identificar posibles vulnerabilidades en el código.
- **Pruebas de seguridad de integración (Integration):** en esta capa intermedia se realizan pruebas de seguridad durante la integración de la aplicación en el pipeline de DevOps, para identificar posibles vulnerabilidades y garantizar que los cambios realizados en el código no introduzcan nuevas vulnerabilidades.
- **Pruebas de seguridad en tiempo real (Runtime):** en esta capa se realizan pruebas de seguridad en tiempo de ejecución de la aplicación, mediante técnicas como el monitoreo de actividad sospechosa, el análisis de registros, la detección de vulnerabilidades en tiempo real, etc.

En la siguiente imagen se pueden apreciar las capas y sus distintos componentes:

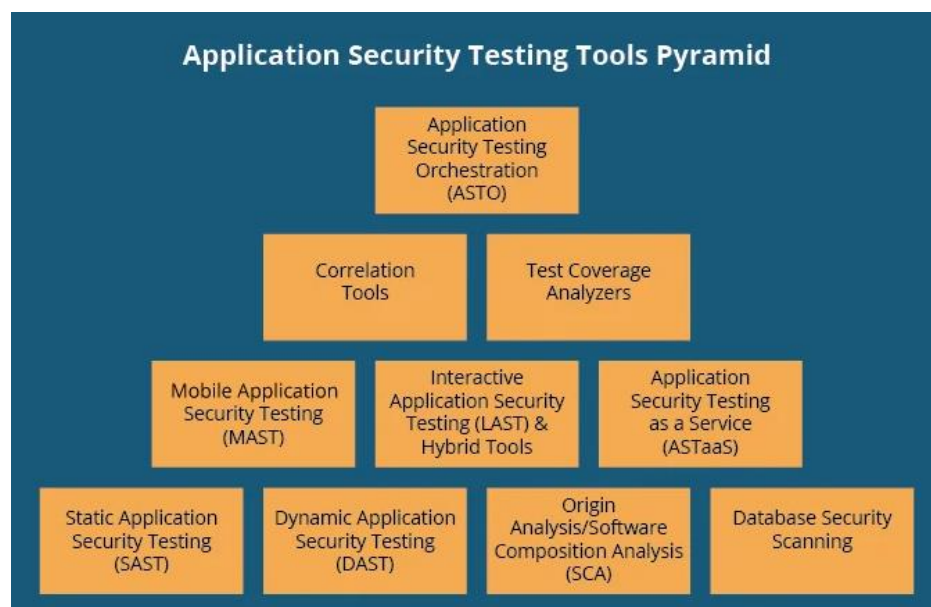


Figura 17: Pirámide AST

Fuente: <https://insights.sei.cmu.edu/blog/10-types-of-application-security-testing-tools-when-and-how-to-use-them/>

4.5 Modelado de amenazas

El Modelado de amenazas (Threat Modeling) se lleva a cabo en las primeras etapas del ciclo de vida del desarrollo de software, como parte del diseño o planificación. El objetivo es identificar posibles

vulnerabilidades o brechas de seguridad en el sistema y evaluar su impacto potencial. El Modelado de amenazas ayuda a los desarrolladores y a los equipos de seguridad a comprender cómo un atacante podría aprovechar una vulnerabilidad para acceder, dañar o robar información del sistema.

Las actividades principales del Modelado de amenazas son las siguientes:

1. **Identificación de activos:** se identifican los activos del sistema que se quieren proteger, como la información, los servidores o los dispositivos.
2. **Identificación de amenazas:** se identifican posibles amenazas o escenarios de amenazas que puedan comprometer los activos del sistema.
3. **Evaluación de riesgos:** se evalúa la probabilidad y el impacto de cada amenaza identificada para priorizar las medidas de seguridad necesarias.
4. **Diseño de medidas de seguridad:** se diseñan medidas de seguridad para mitigar las amenazas identificadas.
5. **Validación:** se valida el modelo de amenazas y las medidas de seguridad diseñadas para asegurarse de que son efectivas y adecuadas para proteger los activos del sistema.

Existen varios enfoques existen varias formas de hacer el modelado de amenazas, entre las principales están:

- **Modelado de amenazas basado en activos:** Este enfoque se centra en identificar las amenazas que pueden afectar a los activos críticos del sistema. El proceso comienza con la identificación de los activos y las amenazas que podrían afectarlos, seguido por la evaluación del riesgo y la selección de las medidas de mitigación adecuadas.
- **Modelado de amenazas basado en datos:** Este enfoque se enfoca en identificar las amenazas que pueden afectar a los datos del sistema. El proceso comienza con la identificación de los datos y las amenazas que podrían afectarlos, seguido por la evaluación del riesgo y la selección de las medidas de mitigación adecuadas.
- **Modelado de amenazas basado en arquitectura:** Este enfoque se enfoca en identificar las amenazas que pueden afectar a la arquitectura del sistema. El proceso comienza con la identificación de los componentes de la arquitectura y las amenazas que podrían afectarlos, seguido por la evaluación del riesgo y la selección de las medidas de mitigación adecuadas.
- **Modelado de amenazas basado en proceso:** Este enfoque se enfoca en identificar las amenazas que pueden afectar a los procesos del sistema. El proceso comienza con la identificación de los procesos y las amenazas que podrían afectarlos, seguido por la evaluación del riesgo y la selección de las medidas de mitigación adecuadas.

Cada enfoque tiene sus propias ventajas y desventajas, y la elección del enfoque adecuado dependerá de los objetivos del proyecto y las características del sistema. En general, se recomienda utilizar una combinación de enfoques para garantizar una cobertura completa de las amenazas potenciales.

Existen varias metodologías para llevar a cabo el modelado de amenazas, siendo las más conocidas STRIDE, DREAD, SEI, OCTAVE y PASTA:

- **STRIDE** es una metodología de Microsoft que se centra en seis tipos de amenazas: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service y Elevation of Privilege.

- **DREAD** es una metodología que evalúa las amenazas en función de cinco factores: Daño potencial, Reproducibilidad, Explotabilidad, Afectados y Descubribilidad.
- **SEI** (Software Engineering Institute) es una organización que ha desarrollado varias metodologías de modelado de amenazas, como **OCTAVE** (Operationally Critical Threat, Asset, and Vulnerability Evaluation) y **PASTA** (Process for Attack Simulation and Threat Analysis). OCTAVE es un marco de trabajo que permite a las organizaciones identificar sus activos críticos, evaluar las amenazas y desarrollar un plan de acción. PASTA, por otro lado, es una metodología que se centra en la simulación de ataques para identificar posibles amenazas y evaluar su riesgo.

Además de estas metodologías formales, también existen herramientas más informales y lúdicas para llevar a cabo el modelado de amenazas. Un ejemplo de ello es **OWASP Cornucopia**, un juego de cartas que ayuda a los equipos de desarrollo a identificar posibles amenazas y mitigaciones.

Las **sesiones informales de pizarra** también pueden ser una forma efectiva de identificar amenazas y riesgos potenciales en un sistema de software.

En la siguiente tabla se pueden ver algunas de las principales herramientas Open Source empleadas en el modelado de amenazas:

Nombre de la herramienta	Descripción	Web de referencia
Threat Dragon	Herramienta de modelado de amenazas con enfoque en la simplicidad y la colaboración, permite crear diagramas de amenazas y documentación relacionada	https://owasp.org/www-project-threat-dragon/
Microsoft Threat Modeling Tool	Herramienta gratuita de Microsoft que ayuda a identificar y mitigar riesgos de seguridad de forma temprana en el ciclo de vida del desarrollo de software	https://www.microsoft.com/en-us/download/details.aspx?id=49168
IriusRisk	Herramienta de modelado de amenazas que se integra con diferentes metodologías de seguridad y proporciona un enfoque de ciclo de vida completo para la gestión de riesgos	https://iriusrisk.com/
Pytm	Biblioteca Python para el modelado de amenazas, ofrece una interfaz gráfica de usuario y permite analizar el impacto de amenazas en diferentes componentes de una aplicación	https://github.com/izar/pytm
P.A.S.T.A. Threat Modeling Tool	Herramienta de modelado de amenazas basada en la metodología P.A.S.T.A. (Process for Attack Simulation and Threat Analysis), proporciona una estructura detallada para el análisis de amenazas y la evaluación de riesgos	https://www.pastaisaca.com/
Threatspec	Herramienta para el modelado de amenazas que permite a los equipos de seguridad crear, compartir y colaborar en modelos de amenazas de manera eficiente.	https://threatspec.org/
Threat playbook	Framework para el modelado de amenazas que proporciona una estructura para el análisis de riesgos y el diseño de soluciones.	https://github.com/threatplaybook/threatplaybook
OWASP Cornucopia	Juego de cartas diseñado para ayudar a los equipos de desarrollo a identificar, discutir y mitigar los riesgos de seguridad en aplicaciones web.	https://owasp.org/www-project-cornucopia/

Tabla 11. Herramientas Open Source para el Threat Modeling

4.6 Análisis estático de la seguridad del código (SAST)

El análisis estático de la seguridad del código (Static Application Security Testing, SAST) es una técnica utilizada en DevSecOps para encontrar vulnerabilidades de seguridad en el código fuente de una aplicación. A diferencia del análisis dinámico, que se realiza en tiempo de ejecución, el análisis estático se realiza sin necesidad de ejecutar el código, sino que se examina el código fuente en busca de patrones o estructuras que puedan indicar una posible vulnerabilidad, defecto o mala práctica.

El análisis SAST es una herramienta muy útil para encontrar vulnerabilidades comunes, como errores de codificación, inyecciones de SQL o Cross-Site Scripting (XSS), entre otras. Además, ayuda a los equipos de desarrollo a encontrar y solucionar problemas de seguridad en el código antes de que se libere al mercado.

Para llevar a cabo un análisis SAST, se utilizan herramientas especializadas que examinan automáticamente el código fuente línea por línea en busca de patrones conocidos de vulnerabilidades. Estas herramientas pueden ser integradas en el pipeline de DevSecOps para realizar análisis continuos a medida que se desarrolla el código. Adicionalmente, es necesario ejecutar inspecciones manuales para descartar falsos positivos o detectar problemas en el código que la herramienta no ha podido encontrar.

A la hora de realizar auditorías de código es vital conocer las principales utilidades que se encuentran disponibles y en todo caso, utilizar varias, ya que es posible que los problemas o defectos que no encuentra una, los pueda detectar otra. Un recurso interesante para conocer las principales herramientas SAST y los lenguajes que soportan es la guía de Source Code Analysis Tools del OWASP [24]. OWASP (Open Web Application Security Project) es un proyecto de código abierto dedicado a determinar y combatir las causas que hacen que el software sea inseguro.

En la siguiente tabla se pueden ver algunas de las principales herramientas Open Source empleadas en el análisis estático de código:

Nombre de la herramienta	Descripción	Web de referencia
SonarQube	Detecta problemas de calidad del código, incluyendo vulnerabilidades de seguridad, errores, y problemas de mantenibilidad. Incluye los lenguajes Java, JavaScript, Python, C++, C# y Go.	https://www.sonarqube.org/
Checkmarx	Analiza el código fuente en busca de vulnerabilidades de seguridad y de cumplimiento de políticas.	https://www.checkmarx.com/
Fortify	Analiza el código fuente en busca de vulnerabilidades de seguridad, y se integra en IDE y sistemas de compilación.	https://www.microfocus.com/en-us/
ESLint	Analiza el código fuente de JavaScript y TypeScript, en busca de problemas de estilo, errores y vulnerabilidades de seguridad.	https://eslint.org/
PMD	Analiza el código fuente de Java, Javascript y XML, en busca de errores, vulnerabilidades de seguridad y problemas de estilo de código.	https://pmd.github.io/
Insider	Herramienta de análisis de código fuente para identificar vulnerabilidades en el código escrito.	https://github.com/insidersec/insider
Retire	Analiza la lista de dependencias de un proyecto y comprueba si hay vulnerabilidades conocidas.	https://github.com/RetireJS/retire.js
NodeJSScan	Herramienta de análisis de seguridad para aplicaciones Node.js.	https://opensecurity.in/nodejsscan/

Dependency-Check	Identifica las dependencias de un proyecto y comprueba si hay vulnerabilidades conocidas.	https://owasp.org/www-project-dependency-check/
npm-audit	Analiza las dependencias de un proyecto de Node.js y busca vulnerabilidades conocidas.	https://docs.npmjs.com/cli/v7/commands/npm-audit
Snyk	Analiza las dependencias de un proyecto y busca vulnerabilidades conocidas en tiempo real.	https://snyk.io/
Auditjs	Analiza la seguridad de las dependencias de un proyecto de Node.js.	https://github.com/audittjs/auditjs
Secure Code Scan	Herramienta de análisis de seguridad de código fuente para aplicaciones .NET.	https://securecodescan.com/
Bandit	Herramienta de análisis de seguridad de código fuente para aplicaciones Python.	https://github.com/PyCQA/bandit
DawnsScanner	Analiza la seguridad de las dependencias de un proyecto Ruby on Rails.	https://rubygems.org/gems/dawnsScanner
Find Security Bugs	Herramienta de análisis de seguridad de código fuente para aplicaciones Java.	http://find-security-bugs.github.io/
SpotBugs	Herramienta de análisis de seguridad de código fuente para aplicaciones Java.	https://spotbugs.github.io/

Tabla 12. Herramientas Open Source para SAST

Dentro del análisis estático del código también se pueden encontrar herramientas de cobertura de código testeado, test-coverage tools, que se utilizan para medir la cobertura de las pruebas en el código y garantizar que todas las funciones y líneas de código estén siendo evaluadas adecuadamente. De esta forma, se busca minimizar los errores y mejorar la calidad del software en términos de seguridad y funcionalidad. Se utilizan por ejemplo cuando el software tiene que cumplir alguna certificación, como por ejemplo en el software embarcable de los aviones.

Algunas de las herramientas más empleadas para el análisis de la cobertura del código son las siguientes:

Nombre de la herramienta	Descripción	Web de referencia
Cobertura	Herramienta de análisis de cobertura de código para aplicaciones Java	https://cobertura.github.io/cobertura/
Coverlet	Framework de medición de cobertura de código para .NET Core	https://github.com/coverlet-coverage/coverlet
gcov	gcov es una herramienta de análisis de cobertura de código que se incluye en la mayoría de los compiladores de C++ GNU. Proporciona información detallada sobre cuántas veces se ha ejecutado cada línea de código, así como cuántas veces se ha ejecutado cada función en el código.	https://gcc.gnu.org/onlinedocs/gcc/Gcov.html
Google Test	Google Test es una biblioteca de pruebas de unidad para C++ que incluye una herramienta de análisis de cobertura de pruebas llamada Gcov. Gcov proporciona información detallada sobre la cobertura de las pruebas de una aplicación C++.	https://github.com/google/googletest
Istanbul	Herramienta de cobertura de código JavaScript que se integra con varios frameworks de testing	https://istanbul.js.org/

Jacoco	Herramienta de análisis de cobertura de código para aplicaciones Java	https://www.jacoco.org/jacoco/
Icov	Icov es una herramienta que proporciona un informe detallado sobre la cobertura de las pruebas de una aplicación C++. Se ejecuta sobre los resultados de la herramienta gcov y produce informes en formato HTML que pueden ser visualizados en un navegador web.	https://github.com/linux-test-project/lcov
OpenCppCoverage	OpenCppCoverage es una herramienta de análisis de cobertura de pruebas de código abierto para C++ en Windows. Proporciona información detallada sobre la cobertura de las pruebas unitarias y de integración en un informe HTML.	https://github.com/OpenCppCoverage/OpenCppCoverage
PHPUnit	Marco de prueba de unidad para PHP que incluye herramientas de análisis de cobertura de código	https://phpunit.de

Tabla 13. Herramientas Open Source para el análisis de cobertura de código

4.7 Análisis de secretos

Actualmente, existe una proliferación no gestionada de secretos o credenciales (como contraseñas, claves de API, tokens, etc.) en diferentes partes del código, sistemas y herramientas utilizados en el proceso de desarrollo y despliegue de aplicaciones. Es lo que se conoce con el término "secret sprawl".

Una mala gestión de los secretos es peligrosa, ya que puede llevar a la exposición accidental de credenciales confidenciales a personas no autorizadas, lo que podría comprometer la seguridad de una aplicación. Por ejemplo, si un desarrollador almacena una clave de API en un archivo de configuración que se incluye en el código fuente y se sube a un repositorio público, cualquier persona que tenga acceso a ese repositorio podría obtener esa clave y utilizarla para fines maliciosos.

Para evitar el "secret sprawl", se utilizan herramientas de gestión de secretos y prácticas de seguridad adecuadas, como el cifrado de secretos, la rotación regular de credenciales y la limitación de acceso a los secretos solo a las personas y sistemas autorizados.

En la siguiente tabla se pueden ver algunas de las principales herramientas Open Source empleadas en el análisis de secretos:

Nombre de la herramienta	Descripción	Web de referencia
Blacklist	Herramienta para analizar el código fuente y detectar patrones de cadenas que se deben evitar	https://github.com/step-ancheg/rust-regex-automata
Confidant	Herramienta para almacenar y gestionar secretos de aplicaciones	https://github.com/lyft/confidant
Detect-secrets	Herramienta de análisis de secretos para evitar que se expongan inadvertidamente en el repositorio	https://github.com/Yelp/detect-secrets
Git Hound	Herramienta de análisis de secretos para Git que busca secretos en los repositorios de código fuente.	https://github.com/ezeckiel/git-hound
GitGuardian	Plataforma de detección de secretos que busca en repositorios de código fuente, sistemas de control de versiones y otros lugares en busca de claves API, contraseñas, tokens y otros secretos.	https://www.gitguardian.com/

Gitleaks	Herramienta de análisis de secretos para Git que busca información confidencial en repositorios y confirmaciones.	https://github.com/zricethezav/gitleaks
Gitleaks-action	Acción de Github para ejecutar Gitleaks en un repositorio y detectar fugas de información en Git	https://github.com/marktplace/actions/gitleaks-action
git-secrets	Herramienta de análisis de secretos para Git que busca patrones específicos en los archivos del repositorio y las confirmaciones.	https://github.com/awslabs/git-secrets
Gittyleaks	Herramienta de análisis de secretos para Git que busca información confidencial en repositorios y confirmaciones.	https://github.com/m3dev/gittyleaks
HashiCorp Vault	Herramienta para almacenar y gestionar secretos de aplicaciones en una bóveda centralizada	https://www.vaultproject.io/
Repo Supervisor	Herramienta de análisis de secretos para Git que busca contraseñas y claves de API en repositorios y archivos.	https://github.com/auth0/repo-supervisor
Secretlint	Herramienta de análisis de secretos para detectar información confidencial y datos personales en el código fuente	https://github.com/secretlint/secretlint
Shhgit	Busca secretos almacenados en repositorios públicos de Git	https://github.com/eth0izzle/shhgit
Talisman	Herramienta de análisis de secretos para Git que evita que se envíen datos sensibles al repositorio.	https://thoughtworks.github.io/talisman/
TruffleHog	Herramienta de análisis de secretos para Git que busca contraseñas, claves API y otros secretos en el historial de confirmaciones y los archivos del repositorio.	https://github.com/trufflesecurity/truffleHog

Tabla 14. Herramientas Open Source para el análisis de secretos

4.8 Análisis de componentes del software (SCA)

El análisis de componentes de software (Software Composition Analysis, SCA) se centra en detectar vulnerabilidades en las bibliotecas y componentes de software de terceros que se utilizan en un desarrollo software. Estas vulnerabilidades pueden ser explotadas por atacantes para comprometer la seguridad de una aplicación, incluso si el código interno de la aplicación en sí es seguro.

El análisis de componentes de software se realiza escaneando los componentes de software de terceros en un desarrollo software para identificar cualquier vulnerabilidad conocida o debilidad en la versión utilizada. Los resultados de este análisis se comparan con una base de datos de vulnerabilidades conocidas para determinar si existe algún riesgo de seguridad. Si se detecta una vulnerabilidad, se toman medidas para remediar el problema, como actualizar el componente a una versión más segura o buscar una alternativa más segura.

Es importante tener en cuenta que el análisis de componentes de software es una práctica crítica en DevSecOps ya que muchos atacantes buscan explotar las vulnerabilidades en los componentes de software de terceros en lugar de intentar atacar directamente el código interno de la aplicación. Al implementar un análisis de componentes de software efectivo, se pueden detectar y remediar rápidamente las vulnerabilidades en los componentes de software de terceros, lo que ayuda a mejorar la seguridad global de la aplicación.

En la siguiente tabla se pueden ver algunas de las principales herramientas Open Source empleadas en el análisis de componentes de software:

Nombre de la herramienta	Descripción	Web de referencia
Dependency-Check	Analiza las dependencias de software de un proyecto en busca de vulnerabilidades conocidas en componentes de terceros.	https://owasp.org/www-project-dependency-check/
FOSSA	Identifica y gestiona las dependencias de código abierto en proyectos de software. Proporciona información sobre las licencias de los componentes.	https://fossa.com/
GitLab Dependency Scanning	Analiza las dependencias de un proyecto de GitLab en busca de vulnerabilidades conocidas. Puede integrarse con varios proveedores de escaneo de vulnerabilidades.	https://docs.gitlab.com/ee/user/application_security/dependency_scanning/
Greenkeeper	Monitoriza las dependencias de un proyecto de JavaScript y notifica al usuario de actualizaciones disponibles.	https://greenkeeper.io/
Hakiri	Analiza las dependencias de un proyecto Ruby en busca de vulnerabilidades conocidas. Proporciona parches y sugerencias para corregir las vulnerabilidades.	https://hakiri.io/
JFrog Xray	Analiza los componentes de software de un proyecto en busca de vulnerabilidades conocidas. Proporciona información sobre las licencias de los componentes y la conformidad con las políticas de seguridad.	https://jfrog.com/xray/
Nexus Repository OSS	Gestiona los componentes de software de un proyecto y analiza las dependencias en busca de vulnerabilidades conocidas.	https://www.sonatype.com/nexus/repository-oss
Node Security Platform	Analiza las dependencias de un proyecto Node.js en busca de vulnerabilidades conocidas. Proporciona parches y sugerencias para corregir las vulnerabilidades.	https://nodesecurity.io/
OWASP Dependency-Track	Gestiona las dependencias de software de un proyecto y analiza los componentes en busca de vulnerabilidades conocidas. Proporciona información sobre las licencias de los componentes y la conformidad con las políticas de seguridad.	https://dependencytrack.org/
Retire.js	Analiza las dependencias de un proyecto de JavaScript en busca de vulnerabilidades conocidas en componentes de terceros.	https://github.com/RetireJS/retire.js/
Snyk	Analiza las dependencias de un proyecto en busca de vulnerabilidades conocidas y proporciona parches y sugerencias para corregir las vulnerabilidades. También proporciona información sobre las licencias de los componentes.	https://snyk.io/
Sonatype Nexus IQ	Analiza los componentes de software de un proyecto en busca de vulnerabilidades conocidas y proporciona información sobre las licencias de los componentes y la conformidad con las políticas de seguridad.	https://www.sonatype.com/nexus-iq-server
Trivy	Analiza los contenedores de Docker y las imágenes de Kubernetes en busca de vulnerabilidades conocidas en los componentes de software.	https://github.com/aquasecurity/trivy
WhiteSource Bolt	Analiza las dependencias de un proyecto en busca de vulnerabilidades	https://whitesource.atlassian.net/

Tabla 15. Herramientas Open Source para SCA

4.9 Análisis dinámico de la seguridad de las aplicaciones (DAST)

El análisis dinámico de aplicaciones (Dynamic Application Security Testing, DAST) es una técnica de seguridad utilizada en DevSecOps para identificar vulnerabilidades de seguridad en una aplicación en tiempo de ejecución, mediante la realización de pruebas que simulan ataques externos a la aplicación.

Esta técnica es complementaria al análisis estático del código (SAST), que se enfoca en la revisión del código fuente o binario de la aplicación para identificar posibles vulnerabilidades y al resto de pruebas ejecutadas en las etapas del S-SDLC anteriores a la de despliegue. El objetivo del análisis DAST es detectar vulnerabilidades que solo son visibles en tiempo de ejecución, como problemas de configuración, autenticación, autorización, inyección de código, entre otros.

Este tipo de pruebas son conocidas como “black-box” ya que se parte de la premisa que se desconoce el funcionamiento interno de la aplicación o no se tiene control sobre él, es decir, no se puede depurar o interactuar con el código en funcionamiento. Normalmente, este tipo de pruebas se realizan de forma automatizada utilizando herramientas especializadas y también de forma manual por parte de pentesters o auditores de seguridad.

Para llevar a cabo el análisis DAST, se utilizan herramientas que simulan ataques en la aplicación, enviando solicitudes a la aplicación y analizando las respuestas recibidas. Estas herramientas utilizan diferentes técnicas para descubrir vulnerabilidades, como la inyección de datos, la fuerza bruta, la suplantación de identidad, entre otras. Una vez que se detecta una vulnerabilidad, la herramienta genera un informe detallado que incluye la ubicación de la vulnerabilidad, una descripción de la misma y una recomendación para corregirla.

En la siguiente tabla se pueden ver algunas de las principales herramientas Open Source empleadas en el análisis DAST:

Nombre de la herramienta	Descripción	Web de referencia
Arachni	Herramienta de seguridad web que permite identificar vulnerabilidades de seguridad en aplicaciones web	http://www.arachni-scanner.com/
Grabber	Herramienta de escaneo de sitios web que permite identificar vulnerabilidades comunes en aplicaciones web	https://github.com/No-Github/grabber
Nikto	Herramienta de escaneo de servidores web que realiza pruebas de vulnerabilidades comunes en aplicaciones web	https://github.com/sullo/nikto
Nmap	Herramienta de escaneo de puertos que permite identificar puertos abiertos y servicios en un servidor web	https://nmap.org/
Nuclei	Herramienta para realizar pruebas de seguridad dinámicas (DAST) en aplicaciones web y servicios. Utiliza un enfoque basado en plantillas para definir los escenarios de prueba. Proporciona una amplia biblioteca de plantillas predefinidas para escanear y detectar vulnerabilidades comunes, como inyecciones de SQL, XSS (Cross-Site Scripting), desbordamientos de búfer, entre otros.	https://nuclei.projectdiscovery.io/
OpenVAS	Escáner de vulnerabilidades de red que busca debilidades en sistemas y aplicaciones en la red	http://www.openvas.org/
OWASP ZAP	Aplicación web de código abierto que realiza pruebas de seguridad automáticas en aplicaciones web y permite encontrar vulnerabilidades	https://www.zaproxy.org/
Skipfish	Herramienta de escaneo de sitios web que identifica vulnerabilidades de seguridad en aplicaciones web	https://github.com/spinkham/skipfish

Nombre de la herramienta	Descripción	Web de referencia
Vega	Herramienta de análisis de vulnerabilidades de seguridad en aplicaciones web que realiza pruebas de seguridad automáticas	https://subgraph.com/vega/
Wapiti	Herramienta de análisis de seguridad web que realiza pruebas de seguridad en aplicaciones web en busca de vulnerabilidades conocidas	https://wapiti.sourceforge.io/
WebScarab	Herramienta de seguridad web que permite analizar el tráfico HTTP y HTTPS entre un navegador web y un servidor web para identificar posibles vulnerabilidades	http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project

Tabla 16. Herramientas Open Source para DAST

4.10 Análisis de seguridad de los contenedores

El análisis de seguridad de los contenedores es una técnica que se utiliza en DevSecOps para evaluar la seguridad de los contenedores que se utilizan en el desarrollo de aplicaciones. Los contenedores son entornos de software aislados que permiten que las aplicaciones se ejecuten de manera uniforme en diferentes sistemas operativos y entornos. Sin embargo, estos contenedores pueden presentar vulnerabilidades de seguridad que pueden ser explotadas por atacantes externos o internos.

El análisis de seguridad de los contenedores implica el escaneo de los contenedores para identificar vulnerabilidades y problemas de seguridad en los componentes del sistema, incluyendo las bibliotecas de software, los sistemas operativos y las configuraciones de red. Esto permite a los desarrolladores y equipos de seguridad identificar y corregir las vulnerabilidades antes de que sean explotadas por atacantes.

Las herramientas de análisis de seguridad de contenedores utilizan técnicas como el análisis de vulnerabilidades, el análisis de dependencias de software y el escaneo de puertos para identificar posibles problemas de seguridad. Algunas herramientas también proporcionan informes detallados y recomendaciones para corregir los problemas encontrados.

En la siguiente tabla se pueden ver algunas de las principales herramientas Open Source empleadas en el análisis de seguridad de los contenedores:

Nombre de la herramienta	Descripción	Web de referencia
Anchore Engine	Herramienta de análisis de imágenes de contenedores para detectar vulnerabilidades, malwares y configuraciones erróneas.	https://anchore.com/
Clair	Herramienta de análisis de imágenes de contenedores para detectar vulnerabilidades de seguridad.	https://github.com/quay/clair
Falco	Herramienta de seguridad de contenedores que detecta comportamientos anómalos y amenazas en tiempo real.	https://falco.org/
Grafeas	Herramienta de análisis de imágenes de contenedores que almacena metadatos y genera una historia de auditoría de todas las imágenes utilizadas.	https://grafeas.io/
Kube-hunter	Herramienta de análisis de seguridad para entornos Kubernetes que busca vulnerabilidades en el clúster y en las aplicaciones.	https://github.com/aquasecurity/kube-hunter

OpenSCAP	Herramienta de análisis y validación de conformidad de la configuración de los sistemas y de los contenedores.	https://www.open-scap.org/
OSSEC	Herramienta de seguridad de host que incluye módulos para la monitorización de contenedores y la detección de ataques.	https://www.ossec.net/
Sysdig Secure	Herramienta de seguridad para contenedores que detecta vulnerabilidades y amenazas en tiempo real, y genera informes de cumplimiento normativo.	https://sysdig.com/products/secure/
Trivy	Herramienta de análisis de imágenes de contenedores que busca vulnerabilidades de seguridad en los paquetes de software y en los sistemas operativos.	https://github.com/aquasecurity/trivy
Twistlock	Plataforma de seguridad para contenedores que incluye análisis de vulnerabilidades, detección de amenazas, gestión de políticas y cumplimiento normativo.	https://www.twistlock.com/

Tabla 17. Herramientas Open Source para el análisis de contenedores

4.11 Análisis de la infraestructura como código (IAC)

Infraestructura como código (IAC) significa tratar la infraestructura de un sistema como si fuera código, lo que implica que todo lo relacionado con la infraestructura, como servidores, redes y almacenamiento, se define y configura mediante código.

En DevSecOps, se analiza la seguridad de la IAC mediante el uso de herramientas que examinan el código utilizado para crear y configurar la infraestructura, para asegurar que la infraestructura utilizada para ejecutar las aplicaciones esté configurada de manera segura y para prevenir problemas de seguridad en la producción, como por ejemplo que no haya vulnerabilidades que puedan ser explotadas por atacantes.

Estas herramientas detectan vulnerabilidades en la configuración de la infraestructura y proporcionan una evaluación de seguridad que puede ayudar a los equipos de desarrollo y operaciones a identificar y corregir problemas de seguridad. Por tanto, deben asegurar que el código utilizado para definir la infraestructura esté libre de errores de configuración, permisos inadecuados, configuraciones vulnerables y otros problemas de seguridad. Algunas de las prácticas comunes en el análisis de seguridad de la IAC incluyen la revisión del código de IAC, la validación de la seguridad de la configuración y la implementación de controles de seguridad automatizados.

Al igual que en el análisis estático de código, las herramientas de análisis de IAC pueden integrarse en los pipelines de CI/CD para proporcionar comentarios inmediatos a los desarrolladores. Esto ayuda a garantizar que se aborden las debilidades de seguridad lo antes posible, antes de que se conviertan en vulnerabilidades explotables.

Estas herramientas son capaces de analizar el código de infraestructura de varias plataformas, incluyendo AWS, Azure y Google Cloud Platform, entre otros. También pueden generar informes detallados sobre los riesgos y las debilidades encontrados, lo que facilita a los equipos de seguridad y a los desarrolladores tomar medidas para corregirlos.

En la siguiente tabla se pueden ver algunas de las principales herramientas Open Source empleadas en el análisis de seguridad de la infraestructura como código:

Nombre de la herramienta	Descripción	Web de referencia
Checkov	Herramienta de análisis estático de código para la evaluación de políticas y vulnerabilidades en archivos de configuración de infraestructura como código (IAC).	https://www.checkov.io/
Cloud Custodian	Herramienta de automatización de cumplimiento y seguridad para infraestructura en la nube que permite programar políticas para la gestión de recursos de AWS, Azure, Google Cloud y Kubernetes.	https://cloudcustodian.io/
Cloudrail	Herramienta de análisis de seguridad de la infraestructura en la nube para AWS, Azure y Google Cloud. Identifica problemas de seguridad y compliance en las configuraciones de la infraestructura y proporciona una solución para solucionarlos.	https://www.cloudrail.com/
Pre-commit	Herramienta que permite ejecutar scripts de validación y pruebas antes de hacer un commit. Puede ser utilizado para analizar la infraestructura como código y validar la sintaxis y estilo de los archivos.	https://pre-commit.com/
Sentinel	Lenguaje de políticas de HashiCorp que permite definir políticas de seguridad y cumplimiento para infraestructura como código.	https://www.hashicorp.com/products/sentinel
TFLint	Herramienta de análisis de código para Terraform que ayuda a detectar errores y problemas comunes en los archivos de configuración de la infraestructura.	https://github.com/terraform-linters/tflint
Terraform Security Scanner (TSS)	Herramienta de análisis de seguridad de Terraform que busca vulnerabilidades comunes en los archivos de configuración de la infraestructura, como credenciales expuestas y permisos demasiado amplios.	https://github.com/liamg/tfsec
Terrascan	Herramienta de análisis estático de código que busca vulnerabilidades y errores en los archivos de configuración de Terraform.	https://github.com/accurics/terrascan
tfsec	Herramienta de análisis de seguridad para archivos de configuración de Terraform que busca problemas de seguridad y vulnerabilidades comunes.	https://github.com/tfsec/tfsec
Trufflehog	Herramienta de análisis de secreto para el escaneo de código fuente y la búsqueda de secretos y claves API expuestas. Puede ser utilizado para analizar el código de la infraestructura como código en busca de vulnerabilidades.	https://github.com/trufflesecurity/truffleHog

Tabla 18. Herramientas Open Source para el análisis de la IAC

Para realizar el análisis de IAC, se utilizan herramientas de escaneo automatizado que inspeccionan el código de la infraestructura para buscar configuraciones inseguras, errores y otras debilidades. Estas herramientas también pueden verificar que se cumplan los estándares de seguridad y cumplimiento, como PCI-DSS, HIPAA o SOX. Este proceso de verificación de cumplimiento de estándares se suele denominar Compliance as Code.

En la siguiente tabla se pueden ver algunas de las principales herramientas Open Source empleadas en el análisis de seguridad de la Compliance as Code:

Nombre de la herramienta	Descripción	Web de referencia
Chef Inspec	Permite escribir tests que evalúan la configuración de la infraestructura, y además verifica el cumplimiento de las políticas y estándares de seguridad.	https://www.inspec.io/
DevSec Hardening Framework	Proporciona un conjunto de perfiles predefinidos para verificar la configuración de sistemas y aplicaciones de acuerdo con diversas guías de seguridad.	https://dev-sec.io/
Kitchen CI	Es una herramienta de automatización que permite realizar pruebas de integración y aceptación para verificar que la infraestructura definida en el código se pueda implementar correctamente en el entorno de producción.	https://kitchen.ci/
OpenSCAP	Es una herramienta de análisis de cumplimiento que verifica si los sistemas cumplen con diversas normas de seguridad, como CIS, NIST, PCI-DSS, etc.	https://www.open-scap.org/
Serverspec	Permite escribir tests que verifican la configuración de la infraestructura, incluyendo la configuración del sistema operativo y las aplicaciones.	https://serverspec.org/

Tabla 19. Herramientas Open Source para el análisis de la Compliance as Code

4.12 Gestión de vulnerabilidades

La gestión de vulnerabilidades en DevSecOps se refiere a la identificación, evaluación y mitigación de las posibles vulnerabilidades que puedan afectar la seguridad de los sistemas y aplicaciones de una organización.

En este contexto, DevSecOps implica integrar la gestión de vulnerabilidades en los procesos de desarrollo, integración y despliegue de software, de tal manera que se puedan detectar y abordar las vulnerabilidades lo antes posible.

Para lograr esto, es importante contar con herramientas y técnicas que permitan la identificación de vulnerabilidades en todas las fases del ciclo de vida del software, incluyendo pruebas de penetración, análisis estático de código, análisis dinámico de aplicaciones, entre otros.

También es importante realizar el "False Positive Analysis" (Análisis de Falsos Positivos, FPA) que se refiere a la tarea de analizar los resultados de una herramienta de escaneo de vulnerabilidades para identificar y eliminar falsos positivos.

Los falsos positivos son resultados de escaneo que indican la existencia de una vulnerabilidad que en realidad no existe o que no es explotable. Estos pueden ser causados por diversas razones, como configuraciones inusuales del sistema, cambios recientes en el código o problemas con la propia herramienta de escaneo.

El FPA es importante porque la eliminación de falsos positivos permite a los equipos de seguridad centrarse en las vulnerabilidades reales y críticas en lugar de desperdiciar tiempo y recursos en falsos positivos.

Es muy importante tener una cultura de seguridad en la organización, en la que se fomente la colaboración entre los equipos de desarrollo, seguridad y operaciones para abordar y mitigar las vulnerabilidades de manera efectiva.

Algunas de las principales herramientas Open Source empleadas en la gestión de vulnerabilidades son las siguientes:

Nombre de la herramienta	Descripción	Web de referencia
Anchore Engine	Herramienta de análisis de vulnerabilidades para contenedores Docker	https://anchore.com/
Clair	Herramienta de análisis de vulnerabilidades para contenedores	https://github.com/quay/clair
GVM (Greenbone Vulnerability Management)	Herramienta de análisis de vulnerabilidades que escanea y detecta vulnerabilidades en sistemas y aplicaciones	https://www.greenbone.net/en/
Lynis	Herramienta de auditoría de seguridad y análisis de vulnerabilidades en sistemas Linux	https://cisofy.com/lynis/
OpenVAS	Herramienta de análisis de vulnerabilidades que escanea y detecta vulnerabilidades en sistemas y aplicaciones	https://www.openvas.org/
OSSEC	Sistema de detección de intrusos (IDS) que monitorea archivos, logs y otros recursos del sistema en busca de posibles ataques	https://www.ossec.net/
OWASP ZAP	Herramienta de análisis de seguridad de aplicaciones web que detecta vulnerabilidades de seguridad en aplicaciones web	https://www.zaproxy.org/
Security Onion	Distribución de Linux que incluye varias herramientas de análisis de seguridad, como Snort, Suricata, Zeek y Wazuh	https://securityonion.net/
Snort	Sistema de prevención de intrusiones (IPS) que monitorea el tráfico de red en busca de posibles ataques	https://www.snort.org/
Wazuh	Herramienta de seguridad que monitorea eventos de seguridad en tiempo real y analiza logs en busca de posibles ataques	https://wazuh.com/

Tabla 20. Herramientas Open Source para la Gestión de vulnerabilidades

5 Aplicación práctica de implementación de DevSecOps

5.1 DevSecOps Maturity Model

El DevSecOps Maturity Model (DSOMM) [25] es un marco de referencia que proporciona una guía estructurada para permitir a las organizaciones evaluar y mejorar gradualmente la incorporación de prácticas de seguridad en sus procesos de desarrollo y operaciones (DevOps).

El objetivo del DevSecOps Maturity Model es ayudar a las organizaciones a evaluar su nivel de madurez en cuanto a la implementación de prácticas de seguridad en su proceso de desarrollo y operaciones.

Proporciona una serie de niveles de madurez que van desde un enfoque inicial hasta un enfoque completamente integrado de seguridad en todas las actividades de DevOps. Cada nivel de madurez representa un conjunto de capacidades y prácticas que deben ser alcanzadas para pasar al siguiente nivel, esto es, proporciona una hoja de ruta para la implementación de DevSecOps en toda la organización, incluyendo cuatro etapas o niveles de transformación: principiante (Level 1), intermedio (Level 2), avanzado (Level 3) y experto (Level 4).

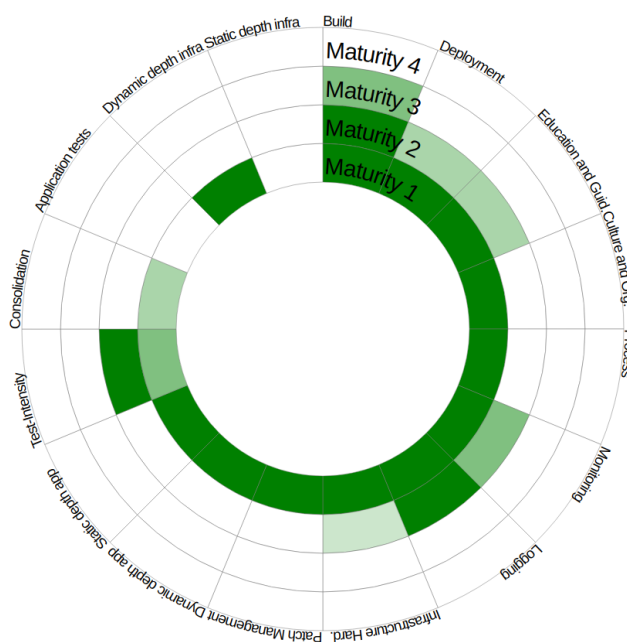


Figura 18. DSOMM Levels

Fuente: <https://owasp.org/www-project-devsecops-maturity-model/>

El modelo se basa en la premisa de que la seguridad no debe ser tratada como una consideración posterior, sino que debe ser una parte integral del proceso de desarrollo y operaciones desde el principio. Esto implica la incorporación de pruebas de seguridad automatizadas, análisis estático de código, escaneo de vulnerabilidades, monitoreo continuo y otras prácticas de seguridad en el flujo de trabajo de DevOps.

Al seguir el DevSecOps Maturity Model, las organizaciones pueden identificar las áreas en las que necesitan mejorar en términos de seguridad, establecer metas realistas y medir su progreso a lo largo del tiempo. Esto ayuda a fomentar una cultura de seguridad en el desarrollo y las operaciones, promoviendo la colaboración entre los equipos y garantizando que la seguridad sea una prioridad en todas las etapas del ciclo de vida del software.

Mover una organización a DevSecOps desde DevOps es un proyecto complicado y en muchos casos supone un desafío importante, se requiere una planificación adecuada para poder implementarlo. La mejor manera de hacerlo es empleando el DevSecOps Maturity Model. Si una organización considera que ya ha alcanzado un determinado nivel de madurez y desea avanzar hacia el siguiente, debe desarrollar todas las actividades requeridas para ese nivel en particular antes de poder abordar el siguiente.

Uno de los objetivos de este proyecto es poder ayudar a las organizaciones a alcanzar un DSOMM Level 1 (DevSecOps Maturity Model Level 1) [26] generando un pipeline de Jenkins completo que pueda ser fácilmente adaptado a sus proyectos.

Consolidar el DSOMM Level 1 exige la ejecución de herramientas de análisis estático (SAST), la detección de secretos y el análisis de componentes de software (SCA), sin ningún cambio en las herramientas o la configuración de las herramientas que se estén utilizando en el desarrollo de software. También se deben ejecutar pruebas con herramientas DAST, pero con su configuración de referencia. La ejecución de estas pruebas de seguridad se puede realizar una vez a la semana o al mes y los informes de las distintas herramientas pueden estar separados. Todas estas pruebas de seguridad se van a incluir en un pipeline completo de Jenkins, en la parte práctica de este proyecto, de manera que cualquier empresa que desarrolle software, pueda adoptarlo en su propio pipeline de Jenkins para alcanzar de una manera sencilla un DSOMM Level 1.

Algunos puntos clave del DSOMM Nivel 1 son:

- No se debe dar por fallida una compilación basada en los resultados del escaneo de seguridad. En este nivel, habrá falsos positivos y se debe evitar la pérdida de la confianza en estas prácticas de seguridad en los equipos de desarrollo.
- La implementación de las herramientas de seguridad y la transferencia de conocimientos a los equipos debe realizarse paulatinamente. Es muy importante que esos equipos tengan experiencia y conocimiento en el área de seguridad, suficientes para ejecutar las herramientas y analizar los resultados.

Es muy importante que los reportes de seguridad generados estén disponibles inmediatamente para los desarrolladores, para que se puedan solucionar los problemas lo antes posible en el SSDLC y no se vayan arrastrando. Cuanto más avanzado esté un desarrollo más costoso será arreglar cualquier problema que se presente, por eso es necesario detectar y solucionar los problemas relativos a la seguridad desde el primer momento.

5.2 Pipelines de Jenkins para DevSecOps

La aplicación seleccionada para analizar estáticamente SAST en el pipeline de Jenkins es VULNADO (Intentionally Vulnerable Java Application) [27] es un servidor web diseñado para enseñar sobre algunas de las 10 principales vulnerabilidades OWASP como por ejemplo SQL Injection, XSS - Cross Site Scripting, SSRF - Server Side Request Forgery y RCE - Remote Code Execution & Reverse Shell. Es un servidor web público que internamente tiene una base de datos Postgres y unas webs internas no accesibles desde el exterior:

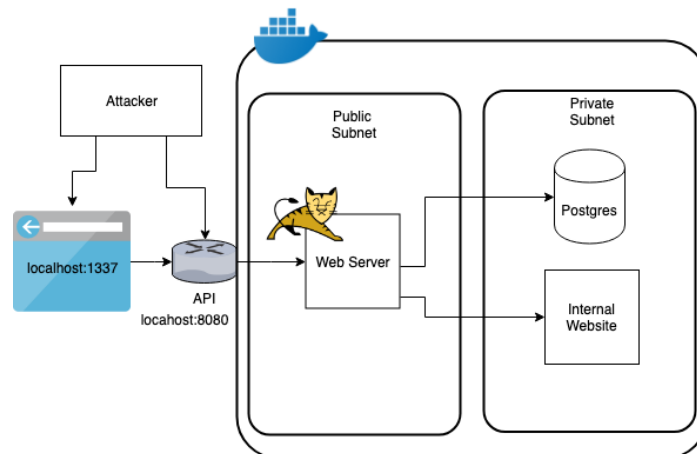


Figura 19. VULNADO Architecture

Fuente: <https://github.com/ScaleSec/vulnado>

Para instalarlo únicamente hay que descargar su código fuente desde su página web [27] y copiarlo en una carpeta, ya que no hace falta ejecutarlo porque solo vamos a ejecutar análisis estáticos (SAST) contra su código fuente.

La aplicación seleccionada para analizar dinámicamente DAST en el pipeline de Jenkins es OWASP Juice-Shop [28], es una página web que simula una tienda online, pero que ha sido desarrollada enteramente para que tenga muchas vulnerabilidades y pueda ser utilizada con fines didácticos:

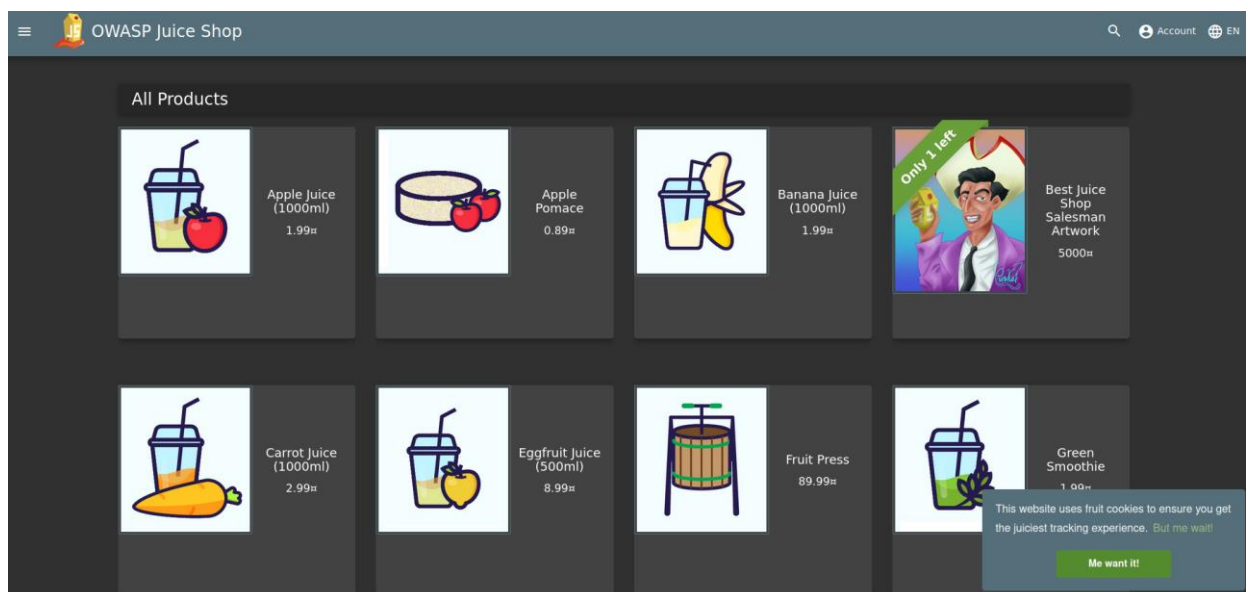


Figura 20. OWASP Juice Shop

Este servidor web sí que debe estar en ejecución para poder realizar pruebas dinámicas (DAST) contra él, para ello lo más sencillo es ejecutarlo desde un contenedor Docker y para ponerlo en ejecución utilizaremos el siguiente comando `sudo docker run -it -d --name jshop -p 0.0.0.0:3000:3000 bkimminich/juice-shop`. Para verificar que está en funcionamiento tan solo hay que abrir una ventana del navegador y poner la siguiente dirección `localhost:3000`.

La Open Web Application Security Project (OWASP) Foundation trabaja para mejorar la seguridad del software a través de sus proyectos de software de código abierto liderados por la comunidad. Juice-shop ha sido creado con fines didácticos para enseñar a detectar algunas de las "Top 10 Web Application Security Risks", que se pueden apreciar en la siguiente imagen:

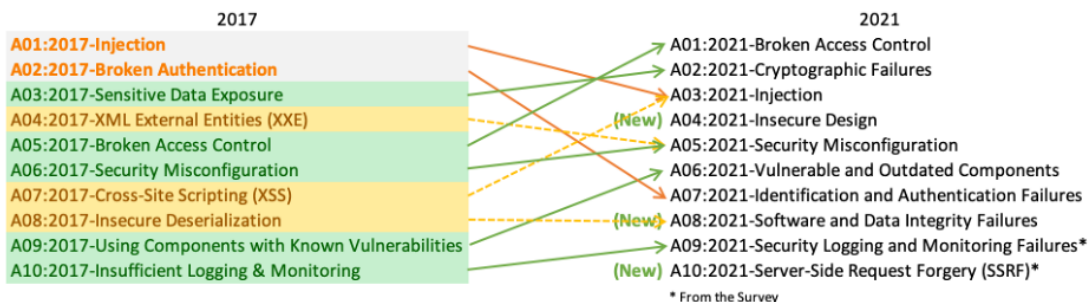
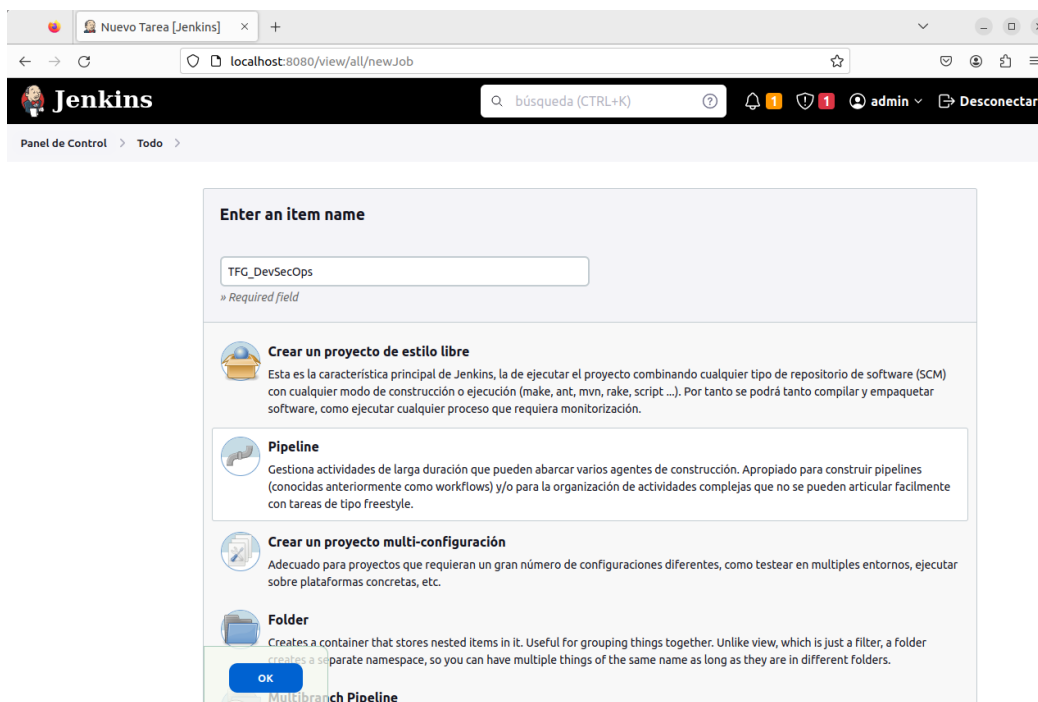


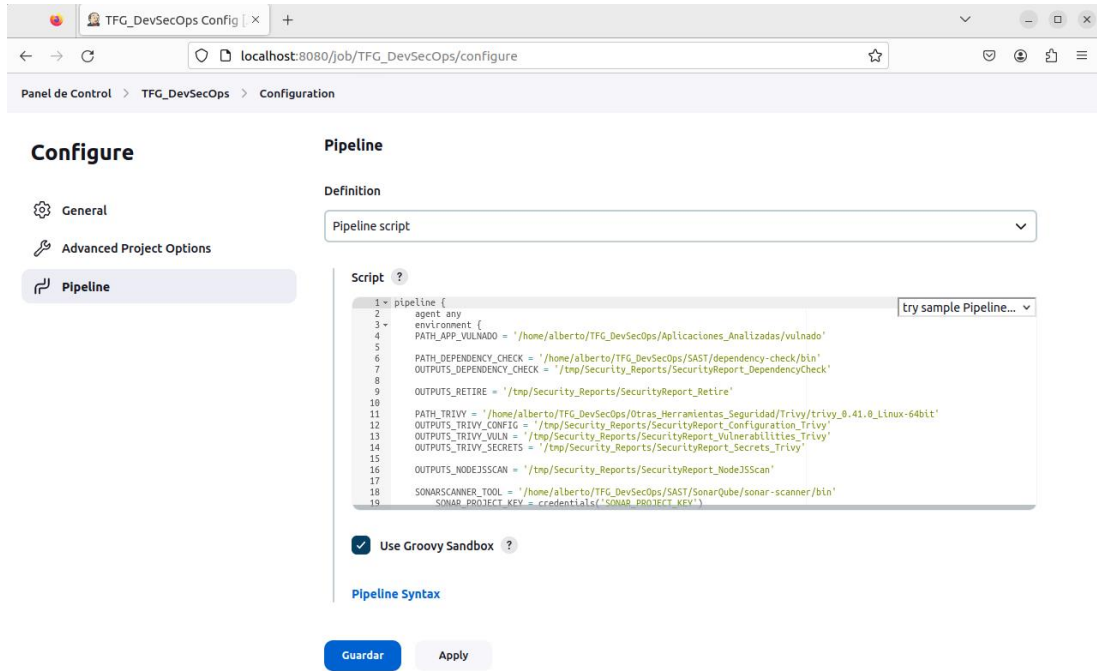
Figura 21. Top 10 Web Application Security Risks 2021

Fuente: <https://owasp.org/www-project-top-ten/>

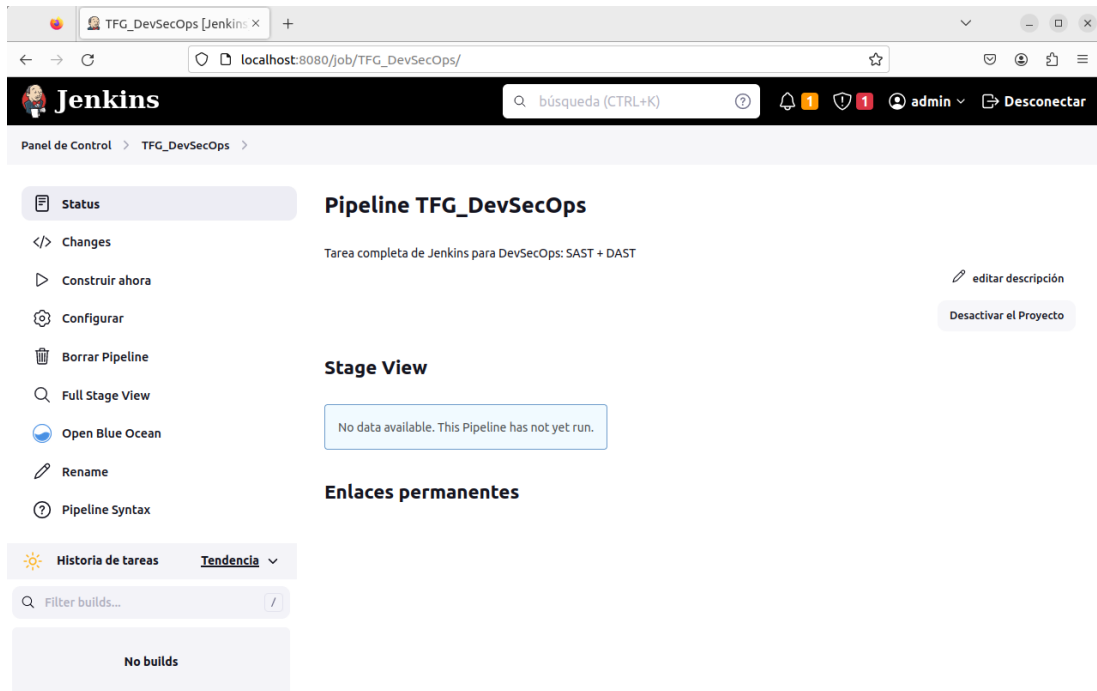
A continuación vamos a crear un pipeline de Jenkins. En la sección 9.1.4 se explica cómo instalar y configurar Jenkins, para usarlo solo tenemos que abrir un navegador web y teclear la dirección "localhost:8080" que nos llevará a la página de inicio de Jenkins, introducimos el usuario y la contraseña y una vez en la pantalla del "Panel de Control" pulsamos sobre "Nueva Tarea", le damos un nombre a la tarea como por ejemplo "TFG_DevSecOps", pulsamos sobre el tipo "Pipeline" y pulsamos sobre el botón "OK":



Dejamos las opciones por defecto y en la zona del "Pipeline" debemos pegar es pipeline completo que hemos desarrollado:



Pulsamos sobre el botón “*Guardar*” y nos redirige al dashboard del pipeline que acabamos de crear. Pulsamos sobre “*Construir ahora*”:

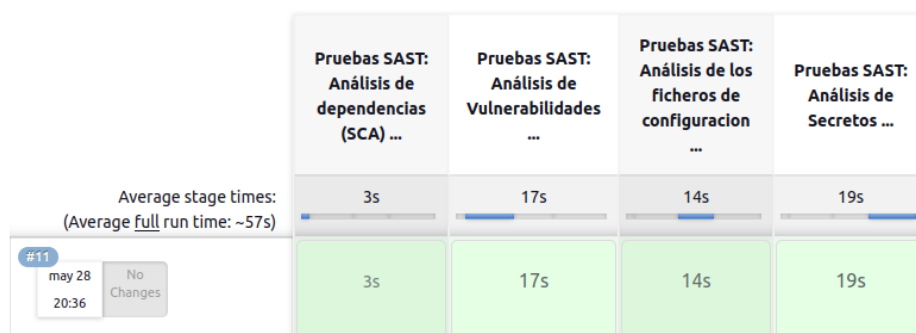


5.3 Ejecución del Pipeline de Jenkins y análisis de resultados

Se han creado dos pipelines de Jenkins independientes, uno para las pruebas estáticas SAST y otro para las pruebas dinámicas DAST, con la intención de que se puedan incorporar más fácilmente en los pipelines de Jenkins de los proyectos de las empresas, al no mezclar el código de ambas partes.

A continuación se muestra el resultado de la ejecución del pipeline SAST en Jenkins:

Stage View



Como se puede apreciar en la imagen anterior, se han ejecutado las siguientes pruebas SAST al proyecto:

- **Análisis de componentes software (SCA):** para identificar y evaluar los componentes de software utilizados en la aplicación en busca de posibles vulnerabilidades conocidas
- **Análisis de vulnerabilidades:** para detectar y remediar vulnerabilidades específicas del código, como errores de programación, malas prácticas y vulnerabilidades conocidas
- **Análisis de los ficheros de configuración:** se revisan los archivos de configuración de la aplicación para detectar configuraciones inseguras o mal configuradas que podrían ser explotadas.
- **Análisis de secretos:** para la identificación y eliminación de información sensible, como contraseñas, claves de acceso o tokens, que pueden estar presentes en el código fuente y representar un riesgo de seguridad si son comprometidos.

El pipeline final de Jenkins de las pruebas SAST que se comentan en el apartado 5.3.1 es el siguiente:

```

1 pipeline {
2     // -----
3     // -- Declaración de los trabajos se pueden ejecutar con cualquier nodo
4     // -----
5     agent any
6
7     // -----
8     // -- Declaración de las variables de entorno
9     // -----
10    environment {
11        PATH_APP_VULNADO = '/home/alberto/TFG_DevSecOps/Aplicaciones_Analizadas/vulnado'
12
13        PATH_DEPENDENCY_CHECK = '/home/alberto/TFG_DevSecOps/SAST/dependency-check/bin'
14        OUTPUTS_DEPENDENCY_CHECK = '/tmp/Security_Reports/'
15
16        OUTPUTS_RETIRE = '/tmp/Security_Reports/SecurityReport_Retire'
17
18        PATH_TRIVY = '/home/alberto/TFG_DevSecOps/Otras_Herramientas_Seguridad/Trivy/trivy_0.41.0_Linux-64bit'
19        OUTPUTS_TRIVY_CONFIG = '/tmp/Security_Reports/SecurityReport_Configuration_Trivy.html'
20        OUTPUTS_TRIVY_VULN = '/tmp/Security_Reports/SecurityReport_Vulnerabilities_Trivy.html'
21        OUTPUTS_TRIVY_SECRETS = '/tmp/Security_Reports/SecurityReport_Secrets_Trivy.html'
22
23        PATH_SONARSCANNER = '/home/alberto/TFG_DevSecOps/SAST/SonarQube/sonar-scanner/bin'
24        SONAR_PROJECT_KEY = credentials('SONAR_PROJECT_KEY')
25        SONAR_TOKEN = credentials('SONAR_TOKEN')
26        SONAR_URL = 'http://localhost:9000'
27    }
28
29    stages {
30        // -----
31        // -- Borrado de los reports de seguridad anteriores
32        // -----
33        stage ('Pruebas SAST: Borrado del contenido del directorio ... '){
34            steps {
35                sh('rm -f /tmp/Security_Reports/*.*')
36            }
37        }
38
39        // -----
40        // -- Ejecución de Pruebas SAST: Análisis de dependencias (SCA) y Vulnerabilidades con Dependency-check
41        // -----
42        stage ('Pruebas SAST: Análisis de Vulnerabilidades con Dependency-check') {
43            steps {
44                sh('PATH_DEPENDENCY_CHECK/dependency-check.sh --scan $PATH_APP_VULNADO -o $OUTPUTS_DEPENDENCY_CHECK')
45            }
46        }
47    }
48

```

```

49 // -----
50 // -- Ejecución de Pruebas SAST: Análisis de Vulnerabilidades con Retire
51 // -----
52 stage ('Pruebas SAST: Análisis de dependencias (SCA) con Retire'){
53   steps {
54     sh('cd $PATH_APP_VULNADO && retire --path $PATH_APP_VULNADO --outputpath $OUTPUTS_RETIRE --exitwith 0')
55     //retire --colors --path /home/alberto/TFG_DevSecOps/Aplicaciones_Analizadas/vulnado --outputformat text --outputpath /tmp/Security_Reports/Retire_Report.txt --exitwith 0
56   }
57 }
58 // -----
59 // -- Ejecución de Pruebas SAST: Análisis de Vulnerabilidades y código fuente con SonarQube
60 // -----
61 stage ('Pruebas SAST: Análisis de Vulnerabilidades y código fuente con SonarQube'){
62   steps {
63     sh('cd $APP_VULNADO && PATH_SONARSCANNER/sonar-scanner -Dsonar.projectKey=$SONAR_PROJECT_KEY -Dsonar.sources=. -Dsonar.java.libraries=. -Dsonar.java.binaries=. -Dsonar.ho
64   }
65 }
66 // -----
67 // -- Ejecución de Pruebas SAST: Análisis de Vulnerabilidades con Trivy
68 // -----
69 stage ('Pruebas SAST: Análisis de Vulnerabilidades con Trivy') {
70   steps {
71     sh('$PATH_TRIVY/trivy filesystem --scanners vuln $PATH_APP_VULNADO -o $OUTPUTS_TRIVY_VULN')
72   }
73 }
74 // -----
75 // -- Ejecución de Pruebas SAST: Análisis de los ficheros de configuración con Trivy
76 // -----
77 stage ('Pruebas SAST: Análisis de los ficheros de configuración con Trivy') {
78   steps {
79     sh('$PATH_TRIVY/trivy filesystem --scanners config $PATH_APP_VULNADO -o $OUTPUTS_TRIVY_CONFIG')
80   }
81 }
82 // -----
83 // -- Ejecución de Pruebas SAST: Análisis de Secretos con Trivy
84 // -----
85 stage ('Pruebas SAST: Análisis de Secretos con Trivy') {
86   steps {
87     sh('$PATH_TRIVY/trivy filesystem --scanners secret $PATH_APP_VULNADO -o $OUTPUTS_TRIVY_SECRETS')
88   }
89 }
90 // -----
91 // -----
92 // -----
93 // -----
94 // -----
95 }

```

Se han realizado las mismas pruebas que en las etapas SAST, pero esta vez contra el servidor web Juice-Shop en ejecución, por ser pruebas DAST. Además, se han empleado algunas de las herramientas más utilizadas actualmente para realizar pruebas DAST como OWASP ZAP Proxy generando un conjunto de pruebas dinámicas muy amplias contra el servidor web bajo análisis.

A continuación se muestra un ejemplo de la ejecución del pipeline DAST en Jenkins:

Stage View



Como se puede apreciar en la imagen anterior, se han ejecutado las siguientes pruebas DAST al proyecto:

- **Análisis dinámicos de seguridad:** con las herramientas Nuclei, Wapiti y ZAP Proxy, los tipos de pruebas que se pueden ejecutar son muy variados y muchas veces son configurables en plantillas.
- **Análisis de vulnerabilidades en contenedores:** para detectar y remediar vulnerabilidades específicas del código, como errores de programación, malas prácticas y vulnerabilidades conocidas dentro de los ficheros del contenedor. Existen unas guías de securización para cada tipo de contenedor, que se deben implementar cada vez que se crea un contenedor nuevo [29].
- **Análisis de los ficheros de configuración en contenedores:** se revisan los archivos de configuración del contenedor para detectar configuraciones inseguras o mal configuradas que podrían ser explotadas.
- **Análisis de secretos en contenedores:** para la identificación y eliminación de información sensible, como contraseñas, claves de acceso o tokens, que pueden estar presentes en el código fuente o los ficheros internos del contenedor y representar un riesgo de seguridad si son comprometidos.

El pipeline completo de Jenkins de las pruebas DAST, que se comentan en el apartado 5.3.2 es el siguiente:

```

1 pipeline {
2   // -----
3   // -- Declaración de los trabajos se pueden ejecutar con cualquier nodo
4   // -----
5   agent any
6
7   // -----
8   // -- Declaración de las variables de entorno
9   // -----
10  environment {
11    APP_JUICESHOP = 'http://localhost:3000/'
12    CONTAINER_JUICESHOP = 'bkimminich/juice-shop'
13
14    OUTPUT_NUCLEI = '/tmp/Security_Reports/reportNuclei'
15    NUCLEI = '/home/alberto/TFG_DevSecOps/DAST/nuclei/nuclei_2.9.4_linux_386'
16
17    OUTPUT_WAPITII = '/tmp/Security_Reports/report.html'
18    WAPITII = '/home/alberto/TFG_DevSecOps/DAST/wapiti/wapiti3-3.1.7/bin/'
19    WAPITII_MODULES = 'xss,sql'
20
21    PATH_TRIVY = '/home/alberto/TFG_DevSecOps/Otras_Herramientas_Seguridad/Trivy/trivy_0.41.0_Linux-64bit'
22    OUTPUTS_TRIVY_CONFIG = '/tmp/Security_Reports/SecurityReport_Configuration_Trivy.html'
23    OUTPUTS_TRIVY_VULN = '/tmp/Security_Reports/SecurityReport_Vulnerabilities_Trivy.html'
24    OUTPUTS_TRIVY_SECRETS = '/tmp/Security_Reports/SecurityReport_Secrets_Trivy.html'
25  }
26
27  parameters {
28    choice choices: ["Baseline", "APIS", "Full"],
29            description: 'Tipo de escaneo',
30            name: 'SCAN_TYPE'
31
32    string defaultValue: "http://192.168.1.60:3000/",
33            description: 'Objetivo del escaneo',
34            name: 'TARGET'
35
36    booleanParam defaultValue: true,
37            description: 'Gen. reporte',
38            name: 'GENERATE_REPORT'
39
40    string defaultValue: "/tmp/reports/reportZAP",
41            description: 'Ruta para guardar el informe',
42            name: 'REPORT_PATH'
43  }
44 }
45
46 stages {
47   stage('Pipeline Info') {
48     steps {
49       script {
50         echo ""
51         Parametros usados:
52         Tipo: ${params.SCAN_TYPE}
53         Objetivo: ${params.TARGET}
54         Informe: ${params.GENERATE_REPORT}
55         ""
56       }
57     }
58   }
59   stage('Pipeline DAST') {
60     parallel {
61       stage('Ejecución de ZAP') {
62         stages {
63           stage('ZAP -> Preparación del entorno Docker') {
64             steps {
65               script {
66                 echo "Pulling de la imagen de ZAP"
67                 sh 'docker pull owasp/zap2docker-stable'
68                 echo "Arrancando el contenedor --> Start"
69                 sh ""
70                 docker run -dt --name owasp \
71                   owasp/zap2docker-stable \
72                   /bin/bash
73                 ""
74               }
75             }
76           }
77           stage('ZAP -> Preparar el directorio de trabajo') {
78             steps {
79               script {
80                 sh ""
81                 docker exec owasp \
82                   mkdir /zap/wrk
83                 ""
84               }
85             }
86           }
87         }
88       }
89     }
90   }
91 }

```

```

87     stage('ZAP -> Escaneando el objetivo usando el contenedor') {
88         steps {
89             script {
90                 scan_type = "${params.SCAN_TYPE}"
91                 echo "---> scan_type: $scan_type"
92                 target = "${params.TARGET}"
93                 if(scan_type == "Baseline"){
94                     sh """
95                         docker exec owasp \
96                         zap-baseline.py \
97                         -t $target \
98                         -x report.xml \
99                         -I
100                    """
101                 }
102                 else if(scan_type == "APIS"){
103                     sh """
104                         docker exec owasp \
105                         zap-api-scan.py \
106                         -t $target \
107                         -x report.xml \
108                         -f openapi \
109                         -I
110                    """
111                 }
112                 else if(scan_type == "Full"){
113                     sh """
114                         docker exec owasp \
115                         zap-full-scan.py \
116                         -t $target \
117                         -x report.xml
118                    """
119                     //--x report-$(date +%d-%b-%Y).xml
120                 }
121                 else{
122                     echo "Se ha especificado un tipo de escaneo invalido..."
123                 }
124             }
125         }
126     }
127
128     stage('ZAP -> Copiando el informe al directorio de trabajo'){
129         steps {
130             script {
131                 sh '''
132                     docker cp owasp:/zap/wrk/report.xml ${WORKSPACE}/report.xml
133                 '''
134             }
135         }
136     }
137
138     post {
139         always {
140             echo "ZAP -> Eliminando el contenedor de ZAP"
141             sh '''
142                 docker stop owasp
143                 docker rm owasp
144             '''
145         }
146     }
147
148
149     // -----
150     // -- Ejecución de Pruebas DAST: Análisis de Vulnerabilidades con Nuclei
151     // -----
152     stage('Pruebas DAST: Análisis de Vulnerabilidades con Nuclei') {
153         steps {
154             sh('${NUCLEI}/nuclei -fr -o $OUTPUT_NUCLEI -u $APP_JUICESHOP')
155         }
156     }
157
158     // -----
159     // -- Pruebas DAST: Análisis de Vulnerabilidades con con Wapiti
160     // -----
161     stage('Pruebas DAST: Análisis de Vulnerabilidades con Wapiti') {
162         steps {
163             sh('${WPAPITI}/wapiti -dr -d 3 -m $WPAPITI_MODULES -o $OUTPUT_WAPITI -u $APP_JUICESHOP')
164         }
165     }
166 }

```



```
167 // -----
168 // -- Ejecución de Pruebas DAST: Análisis de Vulnerabilidades en contenedores con Trivy
169 // -----
170 stage ('Pruebas DAST: Análisis de Vulnerabilidades en contenedores con Trivy') {
171     steps {
172         sh('${PATH_TRIVY}/trivy image --scanners vuln -o $OUTPUTS_TRIVY_VULN $CONTAINER_JUICESHOP')
173     }
174 }
175
176 // -----
177 // -- Ejecución de Pruebas DAST: Análisis de los ficheros de configuración en contenedores con Trivy
178 // -----
179 stage ('Pruebas DAST: Análisis de los ficheros de configuración en contenedores con Trivy') {
180     steps {
181         sh('${PATH_TRIVY}/trivy image --scanners config -o $OUTPUTS_TRIVY_CONFIG $CONTAINER_JUICESHOP')
182     }
183 }
184
185 // -----
186 // -- Ejecución de Pruebas DAST: Análisis de Secretos en contenedores con Trivy
187 // -----
188 stage ('Pruebas DAST: Análisis de Secretos con Trivy') {
189     steps {
190         sh('${PATH_TRIVY}/trivy image --scanners secret -o $OUTPUTS_TRIVY_SECRETS $CONTAINER_JUICESHOP')
191     }
192 }
193 }
194 }
195 }
```

5.3.1 Análisis de los resultados de las pruebas estáticas (SAST)

En las siguientes secciones se muestran los resultados de las pruebas estáticas de seguridad (SAST) sobre la aplicación de prueba Vulnado.

5.3.1.1 Resultados del análisis de componentes de software (SCA) y vulnerabilidades con OWASP Dependency-Check

Esta herramienta analiza las dependencias de nuestro proyecto y recolecta piezas de información de dichas dependencias (las evidencias). Estas evidencias se usan para identificar su Common Platform Enumeration (CPE), que es un identificador que asocia una evidencia a una dependencia concreta. Si se ha identificado algún CPE, se listarán las entradas asociadas al Common Vulnerability and Exposure (CVE).

Para la gestión y consulta de las vulnerabilidades públicas descubiertas en las librerías, existe la Base de datos Nacional de Vulnerabilidades (NVD).

Cuando se ejecuta, Dependency Check se sincroniza automáticamente con los datos de esta base de datos:

```
alberto@DevSecOpsTFG: ~/TFG_DevSecOps/Aplicaciones_Analizadas 139x43
alberto@DevSecOpsTFG:~/TFG_DevSecOps/Aplicaciones_Analizadas$ dependency-check.sh --scan ./vulnado/
[INFO] Checking for updates
[INFO] NVD CVE requires several updates; this could take a couple of minutes.
[INFO] Download Started for NVD CVE - 2002
[INFO] Download Complete for NVD CVE - 2002 (1156 ms)
[INFO] Processing Started for NVD CVE - 2002
[INFO] Download Started for NVD CVE - 2003
[INFO] Download Complete for NVD CVE - 2003 (953 ms)
[INFO] Processing Started for NVD CVE - 2003
[INFO] Download Started for NVD CVE - 2004
[INFO] Download Complete for NVD CVE - 2004 (1138 ms)
[INFO] Download Started for NVD CVE - 2005
[INFO] Processing Complete for NVD CVE - 2003 (9535 ms)
[INFO] Processing Started for NVD CVE - 2004
[INFO] Download Complete for NVD CVE - 2005 (1412 ms)
[INFO] Download Started for NVD CVE - 2006
[INFO] Download Complete for NVD CVE - 2006 (1401 ms)
```

Después de sincronizar los datos de vulnerabilidades realiza el análisis del directorio que le hemos indicado y genera un fichero html con los resultados:

```

alberto@DevSecOpsTFG: ~/TFG_DevSecOps/Aplicaciones_Analizadas 108x21
[INFO] Analysis Started
[INFO] Finished Archive Analyzer (2 seconds)
[INFO] Finished File Name Analyzer (0 seconds)
[INFO] Finished Jar Analyzer (2 seconds)
[INFO] Finished Central Analyzer (17 seconds)
[INFO] Finished Dependency Merging Analyzer (0 seconds)
[INFO] Finished Version Filter Analyzer (0 seconds)
[INFO] Finished Hint Analyzer (0 seconds)
[INFO] Created CPE Index (6 seconds)
[INFO] Finished CPE Analyzer (10 seconds)
[INFO] Finished False Positive Analyzer (0 seconds)
[INFO] Finished NVD CVE Analyzer (1 seconds)
[INFO] Finished RetireJS Analyzer (0 seconds)
[INFO] Finished Sonatype OSS Index Analyzer (1 seconds)
[INFO] Finished Vulnerability Suppression Analyzer (0 seconds)
[INFO] Finished Known Exploited Vulnerability Analyzer (0 seconds)
[INFO] Finished Dependency Bundling Analyzer (0 seconds)
[INFO] Finished Unused Suppression Rule Analyzer (0 seconds)
[INFO] Analysis Complete (37 seconds)
[INFO] Writing report to: /home/alberto/TFG_DevSecOps/Aplicaciones_Analizadas/./dependency-check-report.html
    
```

El fichero que genera, para el caso de la aplicación Vulnado, es enorme, con muchísimas vulnerabilidades y con información exhaustiva de cada una de ellas:

file:///home/alberto/TFG_DevSecOps/Aplicaciones_Analizadas/dependency-check-report.html

DEPENDENCY-CHECK

How to read the report | Suppressing false positives | Getting Help: github issues

Project:

Scan Information (show all):

- dependency-check version: 8.3.1
- Report Generated On: Sat, 17 Jun 2023 10:44:41 +0200
- Dependencies Scanned: 47 (34 unique)
- Vulnerable Dependencies: 16
- Vulnerabilities Found: 165
- Vulnerabilities Suppressed: 0
- ...

Summary

Display: Showing Vulnerable Dependencies (click to show all)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
vulnado-0.0.1-SNAPSHOT.jar:hibernate-validator-6.0.14.Final.jar	cpe:2.3:a:redhat:hibernate_validator:6.0.14:*****	pkg:maven/org.hibernate.validator/hibernate-validator@6.0.14.Final	MEDIUM	2	Highest	31
vulnado-0.0.1-SNAPSHOT.jar:jackson-annotations-2.9.0.jar	cpe:2.3:a:fasterxml:jackson-modules-java8:2.9.0:*****	pkg:maven/com.fasterxml.jackson.core/jackson-annotations@2.9.0	MEDIUM	1	Low	36
vulnado-0.0.1-SNAPSHOT.jar:jackson-databind-2.9.8.jar	cpe:2.3:a:fasterxml:jackson-databind:2.9.8:*****	pkg:maven/com.fasterxml.jackson.core/jackson-databind@2.9.8	CRITICAL	54	Highest	40
vulnado-0.0.1-SNAPSHOT.jar:jsoup-1.8.3.jar	cpe:2.3:a:jsoup:jsoup:1.8.3:*****	pkg:maven/org.jsoup/jsoup@1.8.3	HIGH	2	Highest	34
vulnado-0.0.1-SNAPSHOT.jar:log4j-api-2.11.1.jar	cpe:2.3:a:apache:log4j:2.11.1:*****	pkg:maven/org.apache.logging.log4j/log4j-api@2.11.1	LOW	1	Highest	41
vulnado-0.0.1-SNAPSHOT.jar:logback-core-1.2.3.jar	cpe:2.3:a:qos:logback:1.2.3:*****	pkg:maven/ch.qos.logback/logback-core@1.2.3	MEDIUM	1	Highest	30
vulnado-0.0.1-SNAPSHOT.jar:postgresql-42.2.5.jar	cpe:2.3:a:postgresql:postgresql_driver:42.2.5:*****	pkg:maven/org.postgresql/postgresql@42.2.5	CRITICAL	5	Low	44
vulnado-0.0.1-SNAPSHOT.jar:snakeyaml-1.23.jar	cpe:2.3:a:snakeyaml:project_snakeyaml:1.23:*****	pkg:maven/org.yaml/snakeyaml@1.23	CRITICAL	8	Highest	41

Si pinchamos sobre alguna de las vulnerabilidades nos proporciona mucha información al respecto:

5.3.1.2 Resultados del análisis de vulnerabilidades con Retire.js

Retire.js es un escáner de línea de comandos de vulnerabilidades conocidas, principalmente para aplicaciones Web y para aplicaciones desarrolladas con Node.JS. Si lo ejecutamos en el directorio de la aplicación bajo análisis Vulnado, se puede apreciar en la siguiente imagen que no encuentra ninguna vulnerabilidad, por el hecho de que está desarrollada en Java y no la puede interpretar:

```
alberto@DevSecOpsTFG: ~/TFG_DevSecOps/Aplicaciones_Analizadas/vulnado$ retire
retire.js v4.2.1
Loading from cache: https://raw.githubusercontent.com/RetireJS/retire.js/master/repository/jsrepository.json
```

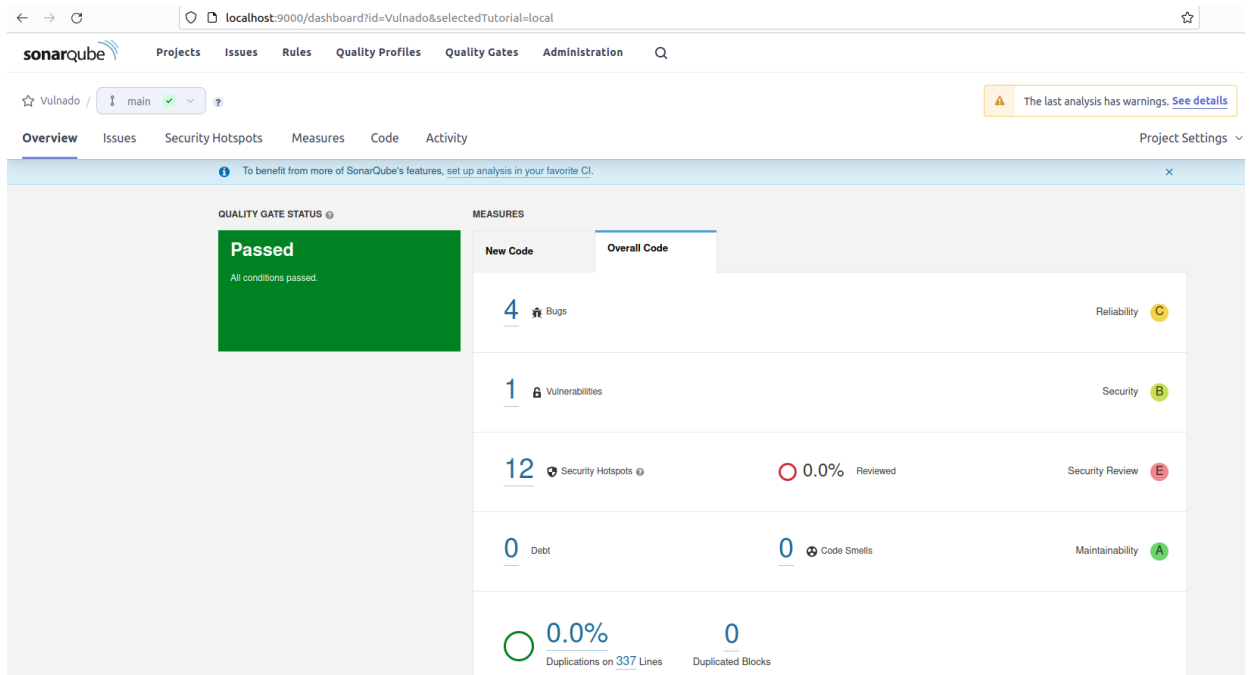
Sin embargo, si lo ejecutamos sobre la otra aplicación que estamos analizando, el servidor web Juice-shop, se puede apreciar que encuentra muchas vulnerabilidades y para cada una de ellas nos muestra su identificador CVE y un link a una página web donde podemos ampliar información sobre esa vulnerabilidad:

```
alberto@DevSecOpsTFG: ~/TFG_DevSecOps/Aplicaciones_Analizadas/juice-shop 108x26
alberto@DevSecOpsTFG:~/TFG_DevSecOps/Aplicaciones_Analizadas/juice-shop$ retire
retire.js v4.2.1
Loading from cache: https://raw.githubusercontent.com/RetireJS/retire.js/master/repository/jsrepository.json
/home/alberto/TFG_DevSecOps/Aplicaciones_Analizadas/juice-shop/node_modules/moment/moment.js
└─ moment.js 2.29.3
moment.js 2.29.3 has known vulnerabilities: severity: high; summary: Regular Expression Denial of Service (ReDoS), Affecting moment package, versions >=2.18.0 <2.29.4, CVE: CVE-2022-31129; https://security.snyk.io/vuln/SNYK-JS-MOMENT-2944238 https://github.com/moment/moment/security/advisories/GHSA-wc69-rhjr-hc9g
/home/alberto/TFG_DevSecOps/Aplicaciones_Analizadas/juice-shop/node_modules/fast.js/dist/bench.js
└─ underscore.js 1.7.0
underscore.js 1.7.0 has known vulnerabilities: severity: high; summary: vulnerable to Arbitrary Code Injection via the template function, CVE: CVE-2021-23358; https://nvd.nist.gov/vuln/detail/CVE-2021-23358
/home/alberto/TFG_DevSecOps/Aplicaciones_Analizadas/juice-shop/node_modules/moment/dist/moment.js
└─ moment.js 2.29.3
moment.js 2.29.3 has known vulnerabilities: severity: high; summary: Regular Expression Denial of Service (ReDoS), Affecting moment package, versions >=2.18.0 <2.29.4, CVE: CVE-2022-31129; https://security.snyk.io/vuln/SNYK-JS-MOMENT-2944238 https://github.com/moment/moment/security/advisories/GHSA-wc69-rhjr-hc9g
/home/alberto/TFG_DevSecOps/Aplicaciones_Analizadas/juice-shop/node_modules/moment/min/moment-with-locales.min.js
└─ moment.js 2.29.3
moment.js 2.29.3 has known vulnerabilities: severity: high; summary: Regular Expression Denial of Service (ReDoS), Affecting moment package, versions >=2.18.0 <2.29.4, CVE: CVE-2022-31129; https://security.snyk.io/vuln/SNYK-JS-MOMENT-2944238 https://github.com/moment/moment/security/advisories/GHSA-wc69-rhjr-hc9g
/home/alberto/TFG_DevSecOps/Aplicaciones_Analizadas/juice-shop/node_modules/moment/min/moment.min.js
└─ moment.js 2.29.3
```

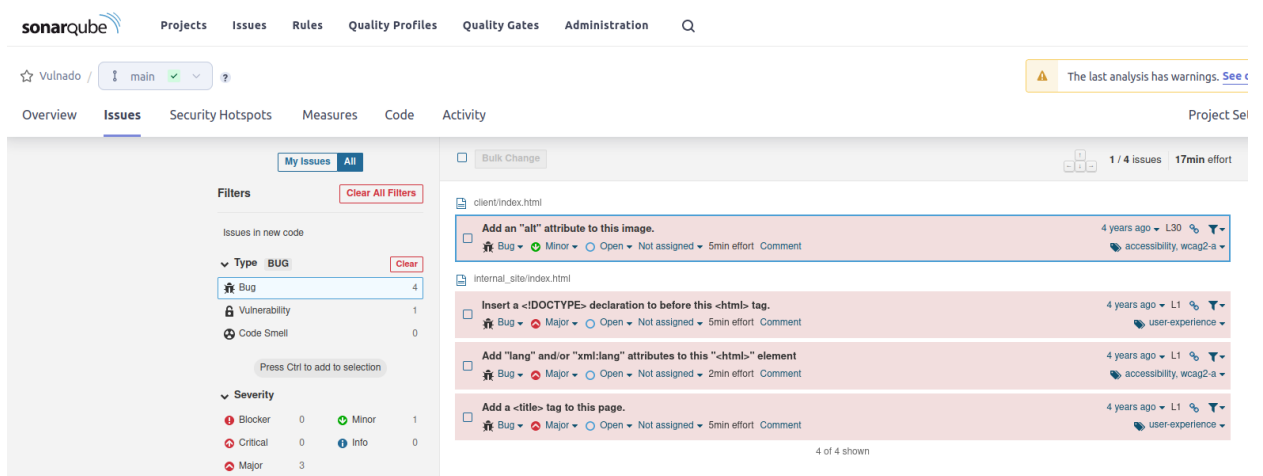
5.3.1.3 Resultados del análisis de vulnerabilidades con SonarQube

SonarQube es una herramienta esencial para mantener y mejorar la calidad del código fuente. Ofrece un análisis exhaustivo del código fuente, detección de problemas y métricas de calidad para facilitar la toma de decisiones informadas y la mejora continua en el desarrollo de software.

Al ejecutar el análisis de SonarQube, se abre una página del navegador mostrando los resultados del análisis:



Se pueden ver los errores y su categorización:



Las vulnerabilidades, que en este caso solo ha detectado una menor:

The screenshot shows the SonarQube interface for a project named 'Vulnado'. The 'Issues' tab is active, displaying a list of issues. A specific issue is highlighted: 'Restrict IP addresses authorized to access administration services.' This issue is categorized as a 'Vulnerability' with a severity of 'Minor' and is 'Open'. It has a 5-minute effort and was reported 4 years ago. The issue is associated with the file 'reverse_shell/main.tf'.

Pinchando sobre ella se muestra el detalle de la vulnerabilidad:

This screenshot provides a detailed view of the vulnerability issue. It shows the issue title 'Restrict IP addresses authorized to access administration services.' and its classification as a 'Vulnerability' with a 'Minor' severity. The issue is linked to the file 'reverse_shell/main.tf'. The interface displays the relevant code snippet from the Terraform file, which defines ingress rules for an AWS VPC. The issue is highlighted in the code, indicating that the ingress rules do not restrict IP addresses.

Nos muestra los principales problemas de seguridad que ha encontrado:

The screenshot shows the 'Security Hotspots' section of the SonarQube interface. It displays a list of security hotspots that need to be reviewed. The current view shows 12 hotspots, with a review priority of 'MEDIUM'. One specific hotspot is highlighted: 'Copying recursively might inadvertently add sensitive data to the container. Make sure it is safe here.' This hotspot is associated with the file 'Dockerfile' and has a status of 'TO REVIEW'. The interface shows the relevant code snippet from the Dockerfile, which includes a 'COPY' command that copies the entire directory into the container.

También nos puede dar métricas del código fuente del proyecto, aunque en este TFG no se van a analizar.

5.3.1.4 Resultados del análisis de vulnerabilidades con Trivy

El stage que se ejecuta del pipeline de Jenkins es el siguiente:

```
// -----
// -- Ejecución de Pruebas SAST: Análisis de Vulnerabilidades con Trivy
// -----
stage ('Pruebas SAST: Análisis de Vulnerabilidades con Trivy') {
  steps {
    sh('$PATH_TRIVY/trivy filesystem --scanners vuln $PATH_APP_VULNADO -o $OUTPUTS_TRIVY_VULN')
  }
}
```

Esto genera un fichero HTML en el directorio definido por la variable OUTPUTS_TRIVY_VULN. Si ejecutamos el mismo comando por línea de comandos el resultado es una lista enorme de vulnerabilidades como las siguientes:

alberto@DevSecOpsTFG: ~/TFG_DevSecOps/Otras_Herramientas_Seguridad/Trivy/trivy_0.41.0_Linux-64bit 139x43					
	The jsoup cleaner may incorrectly sanitize crafted XSS attempts if SafelList.preserveRelativeLinks is...	CVE-2022-36033	MEDIUM		1.15.3
	https://avd.aquasec.com/nvd/cve-2022-36033				
org.postgresql:postgresql	jdbc-postgresql: Unchecked Class Instantiation when providing Plugin Classes	CVE-2022-21724	CRITICAL	42.2.5	42.2.25, 42.3.2
	https://avd.aquasec.com/nvd/cve-2022-21724				
	postgresql-jdbc: Arbitrary File Write Vulnerability	CVE-2022-26520			42.3.3
	https://avd.aquasec.com/nvd/cve-2022-26520				
	postgresql-jdbc: XML external entity (XXE) vulnerability in PgSQLXML	CVE-2020-13692	HIGH		42.2.13
	https://avd.aquasec.com/nvd/cve-2020-13692				

5.3.1.5 Resultados del análisis de los ficheros de configuración con Trivy

El stage que se ejecuta del pipeline de Jenkins es el siguiente:

```
// -----
// -- Ejecución de Pruebas SAST: Análisis de los ficheros de configuracion con Trivy
// -----
stage ('Pruebas SAST: Análisis de los ficheros de configuracion con Trivy') {
  steps {
    sh('$PATH_TRIVY/trivy filesystem --scanners config $PATH_APP_VULNADO -o $OUTPUTS_TRIVY_CONFIG')
  }
}
```

Esto genera un fichero HTML en el directorio definido por la variable OUTPUTS_TRIVY_CONFIG. Si ejecutamos el mismo comando por línea de comandos el resultado es una lista con fallos de seguridad en los ficheros de configuración como los siguientes:


```

alberto@DevSecOpsTFG: ~/TFG_DevSecOps/Otras_Herramientas_Seguridad/Trivy/trivy_0.41.0_Linux-64bit 139x43
LOW: Security group explicitly uses the default description.

Security groups should include a description for auditing purposes.

Simplifies auditing, debugging, and managing security groups.

See https://avd.aquasec.com/misconfig/avd-aws-0099

reverse_shell/main.tf:48-72

48 resource "aws_security_group" "sg" {
49   name           = "tmp_vulnado_rev_shell_sg"
50   vpc_id         = "${aws_vpc.main.id}"
51
52   ingress {
53     protocol = "tcp"
54     from_port = 22
55     to_port   = 22
56     cidr_blocks = ["0.0.0.0/0"]
..
..

CRITICAL: Security group rule allows egress to multiple public internet addresses.

Opening up ports to connect out to the public internet is generally to be avoided. You should restrict access to IP addresses or ranges that are explicitly required where possible.

See https://avd.aquasec.com/misconfig/avd-aws-0104

reverse_shell/main.tf:70

48 resource "aws_security_group" "sg" {
..
70 [   cidr_blocks = ["0.0.0.0/0"]
..
72 }

```

5.3.1.6 Resultados del análisis de los secretos con Trivy

El stage que se ejecuta del pipeline de Jenkins es el siguiente:

```

// -----
// -- Ejecución de Pruebas SAST: Análisis de Secretos con Trivy
// -----
stage ('Pruebas SAST: Análisis de Secretos con Trivy') {
  steps {
    sh('$PATH_TRIVY/trivy filesystem --scanners secret $PATH_APP_VULNADO -o $OUTPUTS_TRIVY_SECRET')
  }
}

```

Esto genera un fichero HTML en el directorio definido por la variable `OUTPUTS_TRIVY_SECRET`. Si ejecutamos el mismo comando por línea de comandos el resultado es una lista con fallos de seguridad en los secretos, pero en este caso en concreto no encuentra ninguno:

```

alberto@DevSecOpsTFG:~/TFG_DevSecOps/Otras_Herramientas_Seguridad/Trivy/trivy_0.41.0_Linux-64bit$ ./trivy filesystem --scanners secret /home/alberto/TFG_DevSecOps/Aplicaciones_Analizadas/vulnado
2023-06-16T21:27:01.690+0200 INFO Secret scanning is enabled
2023-06-16T21:27:01.690+0200 INFO If your scanning is slow, please try '--scanners vuln' to disable secret scanning
2023-06-16T21:27:01.690+0200 INFO Please see also https://aquasecurity.github.io/trivy/v0.41/docs/secret/scanning/#recommendation-for-faster-secret-detection

```

5.3.2 Análisis de los resultados de las pruebas dinámicas (DAST)

En las siguientes secciones se muestran los resultados de las pruebas dinámicas de seguridad (DAST) sobre el servidor web de prueba Juice-shop.

5.3.2.1 Resultados del análisis de vulnerabilidades con Nuclei

Es un analizador de vulnerabilidades de servidores web. Dispone de un conjunto de plantillas que son las que ejecuta contra el servidor, en el momento de la realización de este TFG eran 6004 plantillas.

Se puede ejecutar por línea de comandos y por tanto puede ser integrado fácilmente en un pipeline de Jenkins. Se le pasa la dirección de la página web a analizar y automáticamente empieza a ejecutar pruebas de seguridad contra él.

La potencia de esta herramienta consiste en que tiene más de 6000 plantillas de pruebas disponibles y aumentando cada día, además de que podemos crear nuestras propias plantillas para ajustar los tests a lo que necesitemos.

En el siguiente ejemplo limitamos las pruebas a las plantillas de exposuras, cves y vulnerabilidades, y solo encuentra dos errores:

```
alberto@DevSecOpsTFG: ~/TFG_DevSecOps/DAST/Nuclei/nuclei_2.9.4_linux_386 133x21
alberto@DevSecOpsTFG:~/TFG_DevSecOps/DAST/Nuclei/nuclei_2.9.4_linux_386$ ./nuclei -t exposures -t cves -t vulnerabilities -u http://127.0.0.1:3000/

projectdiscovery.io

[WRN] Found 3 template[s] loaded with deprecated paths, update before v2.9.5 for continued support.
[INF] Current nuclei version: v2.9.4 (outdated)
[INF] Current nuclei-templates version: v9.5.2 (latest)
[INF] New templates added in latest release: 50
[INF] Templates loaded for current scan: 2753
[INF] Targets loaded for current scan: 1
[INF] Templates clustered: 146 (Reduced 151 Requests)
[swagger-api] [http] [info] http://127.0.0.1:3000/api-docs/swagger.json
[prometheus-metrics] [http] [medium] http://127.0.0.1:3000/metrics
```

El stage que se ejecuta del pipeline de Jenkins es el siguiente:

```
// -----
// -- Ejecución de Pruebas DAST: Análisis de Vulnerabilidades con Nuclei
// -----
stage('Pruebas DAST: Análisis de Vulnerabilidades con Nuclei') {
  steps {
    sh('${NUCLEI}/nuclei -fr -o $OUTPUT_NUCLEI -u $APP_JUICESHOP')
  }
}
```

5.3.2.2 Resultados del análisis de vulnerabilidades con Wapiti

Wapiti es un escáner de vulnerabilidades de aplicaciones web. Está desarrollado en Python y se ejecuta por línea de comandos, por lo que es fácil de integrar en un pipeline de Jenkins. Para ejecutarlo, se puede usar la siguiente línea de comandos “`wapiti -d 3 -m all -u http://localhost:3000`”, donde la “-d 3” es para que analice la página principal y los 3 siguientes niveles de jerarquía y el “-m all” es para que utilice todos sus módulos.

Tras ejecutarlo contra el servidor Juice-shop genera una página html con el informe:

```
alberto@DevSecOpsTFG: ~/TFG_DevSecOps/DAST/Wapiti/wapiti3-3.1.7/bin 133x46
alberto@DevSecOpsTFG:~/TFG_DevSecOps/DAST/Wapiti/wapiti3-3.1.7/bin$ ./wapiti -u http://localhost:3000

WAPITI3

Wapiti 3.1.7 (wapiti-scanner.github.io)
[*] Be careful! New moon tonight.
[!] Unable to import module takeover
[!] Unable to import module ssl
[!] Unable to import module log4shell
[!] Unable to find a module named ssl

[*] Launching module xss
[*] 72 pages were previously attacked and will be skipped

[*] Launching module http_headers
Checking X-Frame-Options :
OK
Checking X-Content-Type-Options :
OK

[*] Launching module csp
CSP is not set

[*] Launching module exec

[*] Launching module ssrf

[*] Launching module redirect

[*] Launching module sql
[*] 72 pages were previously attacked and will be skipped

[*] Launching module file

[*] Launching module cookieflags

[*] Launching module permanentxss

[*] Generating report...
A report has been generated in the file /home/alberto/.wapiti/generated_report
Open /home/alberto/.wapiti/generated_report/localhost_3000_06182023_1125.html with a browser to see this report.
```

El informe muestra que solo ha encontrado una vulnerabilidad:



file:///home/alberto/TFG_DevSecOps/DAST/Wapiti/wapiti_report.html

Wapiti vulnerability report

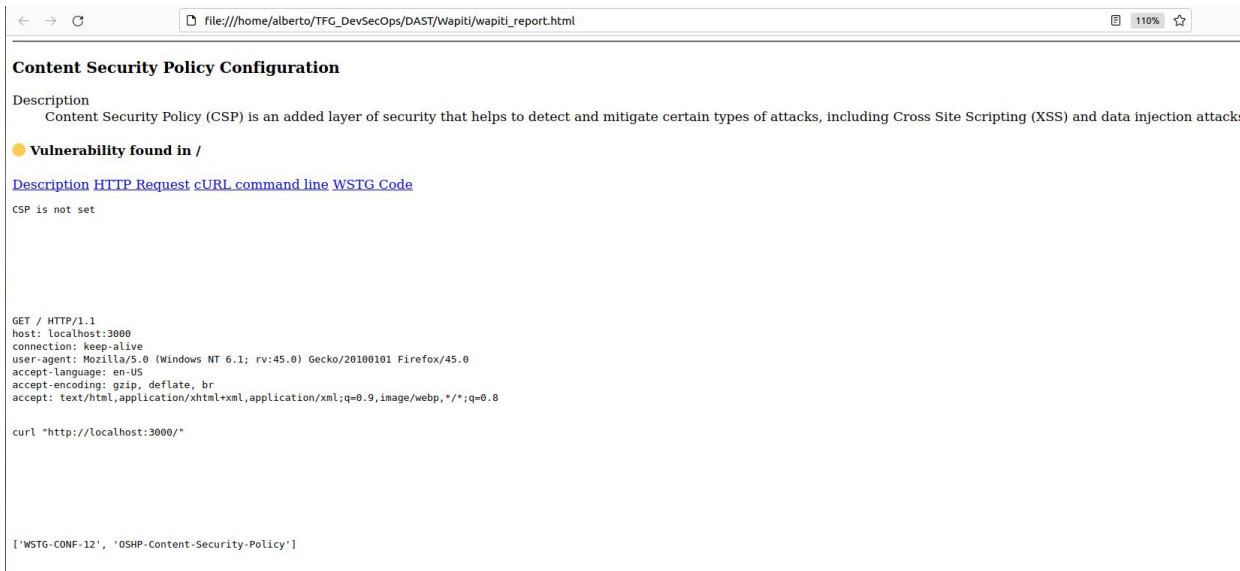
Target: <http://localhost:3000/>

Date of the scan: Sun, 18 Jun 2023 11:25:21 +0000. Scope of the scan: folder. Crawled pages: 72

Summary

Category	Number of vulnerabilities found
Backup file	0
Weak credentials	0
CRLF Injection	0
Content Security Policy Configuration	1
Cross Site Request Forgery	0
Potentially dangerous file	0
Command execution	0
Path Traversal	0
Fingerprint web application framework	0
Fingerprint web server	0
Htaccess Bypass	0
HTTP Secure Headers	0
HttpOnly Flag cookie	0
Log4Shell	0
Open Redirect	0
Reflected Cross Site Scripting	0

Y nos da los detalles de la vulnerabilidad y algunos enlaces web para ampliar la información:



Content Security Policy Configuration

Description
Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks

● **Vulnerability found in /**

[Description](#) [HTTP Request](#) [cURL](#) [command line](#) [WSTG Code](#)

CSP is not set

```
GET / HTTP/1.1
host: localhost:3000
connection: keep-alive
user-agent: Mozilla/5.0 (Windows NT 6.1; rv:45.0) Gecko/20100101 Firefox/45.0
accept-language: en-US
accept-encoding: gzip, deflate, br
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

curl "http://localhost:3000/"
```

['WSTG-CONF-12', 'OSHP-Content-Security-Policy']

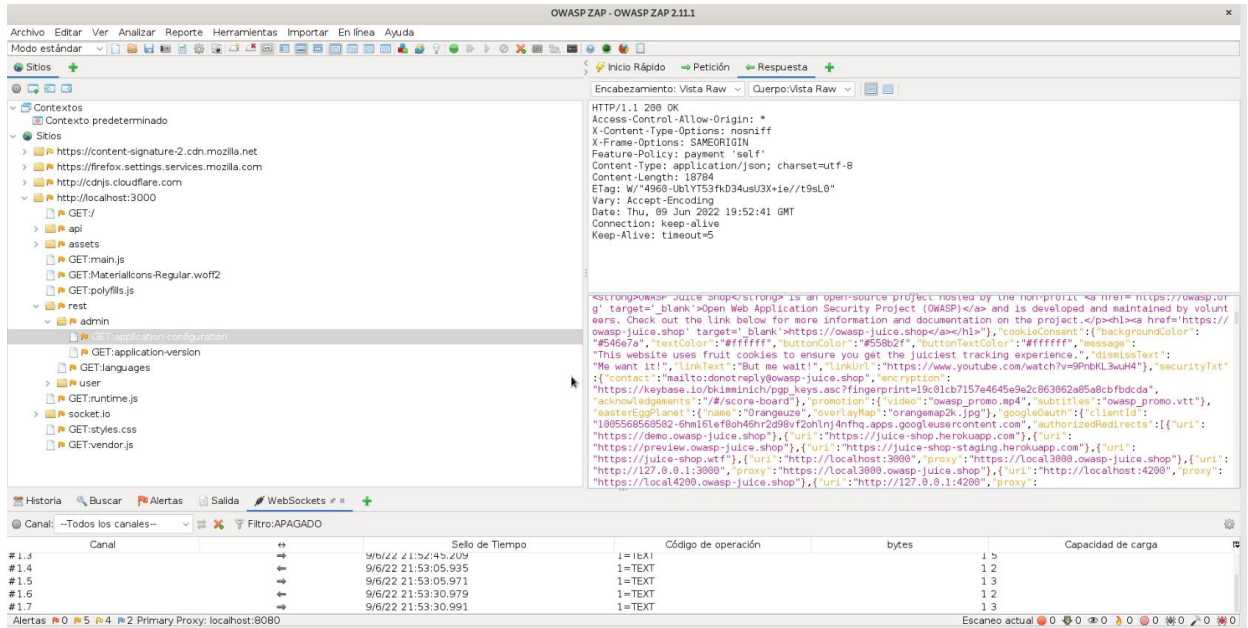
El stage que se ejecuta del pipeline de Jenkins es el siguiente:

```
// -----
// -- Pruebas DAST: Análisis de Vulnerabilidades con con Wapiti
// -----
stage('Pruebas DAST: Análisis de Vulnerabilidades con Wapiti') {
    steps {
        sh('${WAPITI/wapiti -dr -d 3 -m $WAPITI_MODULES -o $OUTPUT_WAPITI -u $APP_JUICESHOP}')
    }
}
```

5.3.2.3 Resultados del análisis de vulnerabilidades con ZAP Proxy

ZAP Proxy es el proyecto insignia de la comunidad OWASP. Es un proxy web, esto es captura las peticiones http y luego las respuestas, de manera que puede analizar ambas. Es un framework completo para pentesting manual y para tests de sitios web de forma automática. Está desarrollado en Java, se puede lanzar por línea de comandos con “*zap.sh*” o integrarlo en un pipeline de Jenkins.

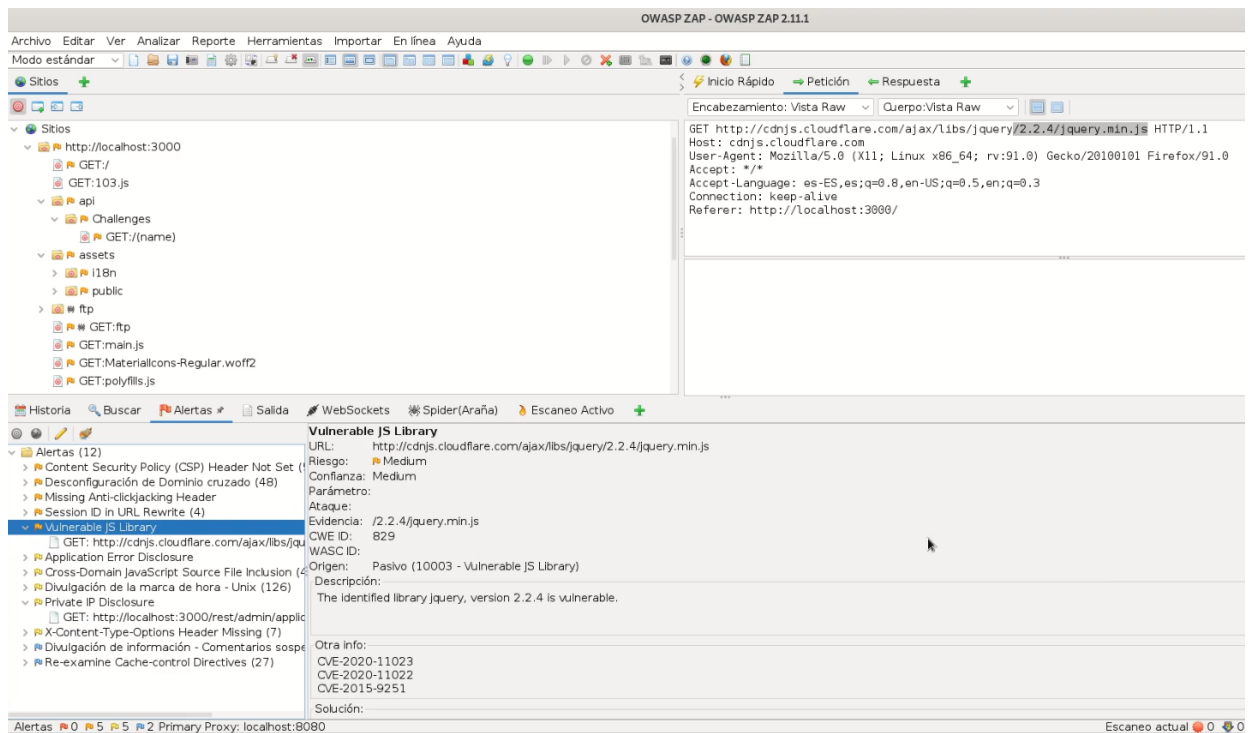
Es una herramienta muy compleja que tiene dos modos de funcionamiento, el modo manual y el automático. En el modo manual ZAP levanta un navegador que gestiona él, de tal manera que puede ir analizando todas las peticiones y respuestas del servidor, es un modo de funcionamiento proxy web y es muy útil para realizar tareas de pentesting.



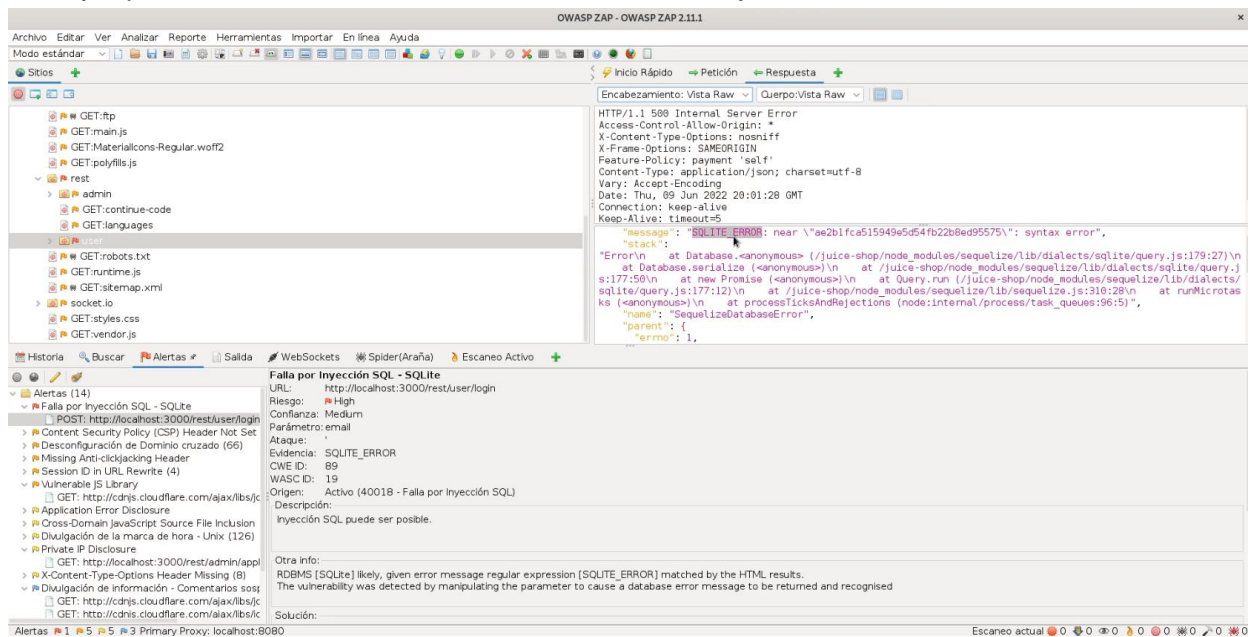
En el modo de funcionamiento automático, introduces la URL destino, pulsas el botón “Atacar” y entonces aplica la política de seguridad y empieza a ejecutar escaneos de vulnerabilidades y ataques a la web de destino. Este modo también se puede implementar en un pipeline de Jenkins.

Tras ejecutarlo contra el servidor Juice-shop genera un report.xml. Para poder ver fácilmente el tipo de información que proporciona vamos a mostrar algunas pantallas con la misma ejecución, pero desde la interfaz gráfica de ZAP.

Por ejemplo, en esta captura se puede apreciar cómo ha detectado vulnerabilidades en la librería Javascript, jquery.min.js:

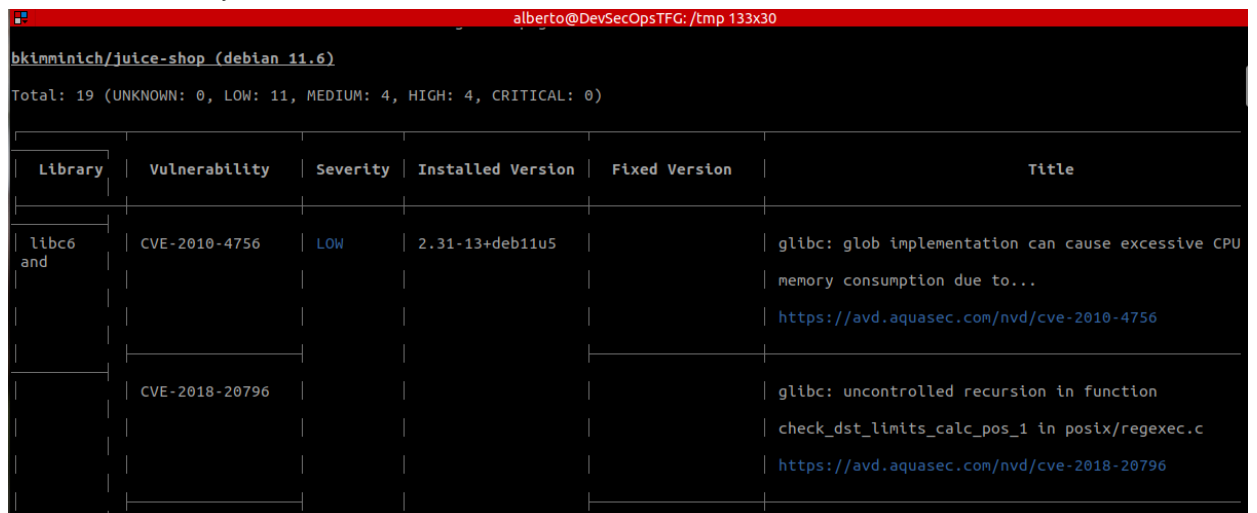


Otro ejemplo es la detección de esta vulnerabilidad de SQL injection:



5.3.2.4 Resultados del análisis de vulnerabilidades en contenedores con Trivy

Trivy también sirve para analizar contenedores, por lo tanto, podemos utilizarlo para analizar las vulnerabilidades de un contenedor Docker, en este caso el de Juice-shop. Lo ejecutamos por línea de comandos (el pipeline de Jenkins ejecuta lo mismo y genera un fichero html con el informe de vulnerabilidades) y este es el reporte que nos genera, donde se pueden ver varias vulnerabilidades, su nivel de criticidad y webs donde obtener información de la vulnerabilidad:



El stage que se ejecuta del pipeline de Jenkins es el siguiente:

```
// -----
// -- Ejecución de Pruebas DAST: Análisis de Vulnerabilidades en contenedores con Trivy
// -----
stage ('Pruebas DAST: Análisis de Vulnerabilidades en contenedores con Trivy') {
  steps {
    sh('${PATH_TRIVY}/trivy image --scanners vuln -o $OUTPUTS_TRIVY_VULN $CONTAINER_JUICESHOP')
  }
}
```

Esto genera un fichero HTML en el directorio definido por la variable OUTPUTS_TRIVY_VULN.

El stage que se ejecuta del pipeline de Jenkins es el siguiente:

```
// -----  
// -- Ejecución de Pruebas DAST: Análisis de Secretos en contenedores con Trivy  
// -----  
stage ('Pruebas DAST: Análisis de Secretos con Trivy') {  
    steps {  
        sh('${PATH_TRIVY}/trivy image --scanners secret -o $OUTPUTS_TRIVY_SECRETS $CONTAINER_JUICESHOP')  
    }  
}
```

Esto genera un fichero HTML en el directorio definido por la variable `OUTPUTS_TRIVY_SECRETS`.

6 Conclusiones

6.1 Conclusiones y lecciones aprendidas

DevSecOps surge como una evolución natural de la metodología DevOps, incorporando un enfoque proactivo en la seguridad durante todo el ciclo de vida del desarrollo de software. Al igual que DevOps, se basa en la colaboración y comunicación entre los equipos de desarrollo y operaciones, pero con el añadido fundamental de integrar la seguridad desde el inicio. DevSecOps reconoce que la seguridad no puede ser un componente adicional al final del proceso, sino que debe ser parte integral de cada etapa, desde el diseño hasta la implementación y el mantenimiento. Su adopción creciente en las empresas de desarrollo de software se debe a los beneficios que aporta en términos de reducción de riesgos y brechas de seguridad, así como en la mejora de la calidad del software entregado.

La gran cantidad de herramientas Open Source que han surgido para trabajar en este tipo de proyectos hace que se puedan implementar en cualquier empresa con un coste de software muy bajo. Sin embargo, la complejidad de los procesos para configurar el ciclo de desarrollo seguro completo (SSDLC), así como la dificultad de involucrar a todas las personas necesarias y de derribar las paredes entre los distintos departamentos hace que sea una tarea muy compleja, pero que merece la pena afrontar por los beneficios que aporta.

Este TFG contribuye a dar una visión general de lo que esta filosofía aporta, tanto a nivel teórico como a nivel práctico. A nivel teórico se hace una breve introducción a todo lo que implica y a nivel práctico se implementa una solución completa de integración y entrega continua con herramientas Open Source de manera que cualquiera pueda implementarla sin tener que invertir en software específico.

A nivel personal, el trabajo realizado durante este TFG ha sido muy satisfactorio, ya que me ha permitido ampliar mis conocimientos sobre DevSecOps y trabajar con herramientas con las que nunca antes había trabajado. Espero poder aplicar todo lo que he aprendido en los proyectos en los que actualmente trabajo.

Para todos los que lean este TFG espero que su lectura sea didáctica y les ayude a adoptar los principios de DevOps y DevSecOps, y que la parte práctica desarrollada les sirva como punto de partida para sus propios procesos de desarrollo internos.

6.2 Objetivos alcanzados

Con la entrega de esta memoria se puede considerar que todos los objetivos propuestos al inicio del TFG, definidos en la sección 1.5, se han cumplido:

- El primer y el segundo objetivos se han cumplido describiendo lo que son DevOps y DevSecOps, así como su historia, sus principios, objetivos, costes, etc., detallados en los capítulos 3 y 4.
- El tercer objetivo se ha cumplido describiendo el estado del arte de las herramientas empleadas en DevOps y DevSecOps, poniendo mayor énfasis en las herramientas Open Source, en los capítulos 3 y 4.
- El cuarto objetivo se ha cumplido con el ejemplo práctico descrito en el capítulo 5 y la guía de instalación del Anexo 9, donde se ha desarrollado un pipeline completo de DevSecOps utilizando Jenkins y algunas de las herramientas Open Source más utilizadas en DevSecOps,

de manera que cualquier empresa pueda utilizarlo, adaptándolo a sus necesidades, y alcance de esa manera un DevSecOps Maturity Model Level 1 (DSOMM Level1).

Respecto al cumplimiento de la planificación inicial, en todo momento se ha seguido la misma, sin desviaciones relevantes que destacar. Por tanto, se ha realizado un trabajo constante durante todo el tiempo de desarrollo del TFG.

6.3 Líneas de trabajo futuras

Como líneas de trabajo futuro, sería recomendable incluir un anexo adicional, similar al Anexo 9.1 de este documento, pero enfocado en la instalación de todos los programas software para el sistema operativo Microsoft Windows 11. Esto permitiría abarcar las mismas herramientas utilizadas en este trabajo, brindando utilidad a aquellos que trabajan con Windows.

Otra línea de trabajo futuro sería generar un entorno de desarrollo completo que incluya un pipeline integral de la parte de DevOps, incluyendo la parte de Deployment. Esto implicaría implementar un proyecto completo de desarrollo de software que pueda ser descargado por cualquier persona, proporcionándole un entorno de referencia listo para comenzar a trabajar en cuestión de minutos, incluyendo un entorno completo de desarrollo con DevOps y que además sea seguro porque incluiría también el pipeline de DevSecOps.

7 Glosario de términos

Término	Significado
API	Interfaz de Programación de Aplicaciones
AST	Application Security Testing
CALMS	Culture, Lean, Automation, Metrics y Sharing
CD	Entrega Continua
CI	Integración Continua
CPE	Common Platform Enumeration
CPU	Central Processing Unit
CVE	Common Vulnerability and Exposure
CyRE	Synopsys Cybersecurity Research Centre
DAST	Dynamic Application Security Testing
DevOps	Development & Operations
DevSecOps	Development & Security & Operations
DSOMM	DevSecOps Maturity Model
FPA	False Positive Analysis
IAC	Infrastructure As Code
NVD	Base de datos Nacional de Vulnerabilidades
OS	Operative System
PEC	Prueba de Evaluación Continua
OSSRA	Open Source Security and Risk Analysis
QA	Calidad
SAST	Static Application Security Testing
SCA	Software Composition Analysis
SCM	Software Code Management
SDLC	Software Development Life Cycle
SSDLC	Secure Software Development Life Cycle
SQL	Structured Query Language
TBD	To Be Defined
TFG	Trabajo Final de Grado
TI	Tecnologías de la Información
UOC	Universitat Oberta de Catalunya
VM	Virtual Machine
WIP	Work In Progress

Tabla 21. Glosario

8 Bibliografía

- [1] Bass, L., Weber, I., & Zhu, L. (2016). DevOps: La Perspectiva de un Arquitecto de Software. Pearson.
- [2] Beizer, B. (2021). Principios de DevSecOps: Asegure y Optimice su Cadena de Entrega de Software. Apress.
- [3] Duffy, M. (2020). DevOps y Seguridad. Packt Publishing.
- [4] Erez, K., & Gonen, O. (2020). DevOpsSec: Asegurando el Software a través de la Entrega Continua. Apress.
- [5] Gruver, M., & Humble, J. (2020). Una Guía Práctica para Entrega Continua: Cómo Asegurar la Entrega Confiable de Software. O'Reilly Media.
- [6] Heynen, S. (2021). Construyendo una Cultura DevSecOps: Una Guía Concisa para Asegurar su Organización a través de la Cultura, Automatización y Métricas. Apress.
- [7] Humble, J., & Farley, D. (2014). Entrega Continua: Cómo Asegurar la Entrega Confiable de Software a Través de la Automatización de Pruebas y Despliegue. Ediciones Deusto
- [8] Jaeger, T., & Breitenbücher, U. (2021). Una arquitectura de referencia para DevSecOps. Anaya Multimedia.
- [9] Kim, G., Behr, K., & Spafford, G. (2019). La cultura DevOps: Un enfoque práctico para desarrollar y mejorar el software. Grupo Anaya Comercial.
- [10] Kim, G., Debois, P., Willis, J., & Humble, J. (2019). El manual DevOps: Cómo crear equipos de alta performance. Ediciones Deusto.
- [11] Kim, G., Cai, Y., Zhu, H., Lu, J., & Xu, S. (2019). DevSecOps: A Collaborative Framework for Secure DevOps. In Proceedings of the 2019 IEEE International Conference on Services Computing (SCC) (pp. 171-174). IEEE.
- [12] Melnyk, I., Pogrebna, G., & Yakovyna, V. (2020). DevSecOps: An Overview of Research and Practices. In Proceedings of the 2020 IEEE International Conference on Advanced Trends in Information Theory and Applications (pp. 131-136). IEEE.
- [13] Palmer, M., & Stöcker, A. (2021). DevSecOps: Integración de la seguridad en los pipelines de entrega continua. Anaya Multimedia.
- [14] Rana, R. (2020). DevSecOps: Creando una Cultura de Seguridad en su Organización. Apress.
- [15] Rose, B. (2020). DevSecOps: Cómo Construir la Seguridad en el Ciclo de Vida de la Entrega de Software. O'Reilly Media.
- [16] Sharma, R., Singh, S., & Kumar, S. (2020). DevSecOps: Una revisión de las mejores prácticas y futuras direcciones de investigación. Editorial Académica Española.

- [18] Shi, L., Zhang, X., & Cui, Y. (2021). Una revisión de DevSecOps: desafíos y soluciones. Editorial Académica Española.
- [18] White, J. (2020). DevSecOps: Construyendo un pipeline de entrega continua seguro. O'Reilly.
- [19] <https://www.devopsdays.org/>, [Fecha de consulta 22/03/2023]
- [20] <https://jbravomontero.wordpress.com/2015/04/29/31-ejemplos-de-arquitectura-para-devops-y-entrega-continua/>, [Fecha de consulta 24/03/2023]
- [21] <https://xebialabs.com/periodic-table-of-devops-tools/>, [Fecha de consulta 27/03/2023]
- [22] <https://www.audea.com/seguridad-en-sdlc/>, [Fecha de consulta 10/04/2023]
- [23] <https://www.izertis.com/es/-/blog/devops-6-fases-clave-para-el-proceso-del-desarrollo-de-software>, [Fecha de consulta 12/04/2023]
- [24] https://owasp.org/www-community/Source_Code_Analysis_Tools, [Fecha de consulta 15/04/2023]
- [25] <https://owasp.org/www-project-devsecops-maturity-model/>, [Fecha de consulta 22/04/2023]
- [26] <https://dsomm.owasp.org/>, [Fecha de consulta 22/04/2023]
- [27] <https://github.com/ScaleSec/vulnado>, [Fecha de consulta 29/04/2023]
- [28] <https://owasp.org/www-project-juice-shop/>, [Fecha de consulta 29/04/2023]
- [29] <https://sysdig.com/blog/dockerfile-best-practices/>, [Fecha de consulta 04/05/2023]
- [30] <https://www.devsecops.org/>, [Fecha de consulta 04/05/2023]
- [31] <https://www.sonarsource.com/products/sonarqube/downloads/>, [Fecha de consulta 11/05/2023]
- [32] <https://docs.sonarqube.org/latest/analyzing-source-code/scanners/sonarscanner/>, [Fecha de consulta 11/05/2023]
- [33] <https://docs.sonarqube.org/latest/setup-and-upgrade/install-the-server/>, [Fecha de consulta 11/05/2023]
- [34] <https://hub.docker.com/>, [Fecha de consulta 12/05/2023]
- [35] <https://owasp.org/www-project-dependency-check/>, [Fecha de consulta 14/05/2023]
- [36] <https://github.com/projectdiscovery/nuclei/releases>, [Fecha de consulta 14/05/2023]
- [37] <https://github.com/wapiti-scanner/wapiti/releases>, [Fecha de consulta 14/05/2023]
- [38] <https://www.zaproxy.org/download/>, [Fecha de consulta 15/05/2023]

9 Anexos

9.1 Instalación de las herramientas DevSecOps

A continuación se van a detallar todas las instrucciones para instalar todas las herramientas de software de seguridad necesarias para ejecutar el pipeline de Jenkins para DevSecOps.

Para la realización del proyecto se ha elegido como herramienta para crear la máquina virtual “Oracle VM VirtualBox” y como sistema operativo a “Ubuntu 22.04.2 LTS”, ya que las versiones LTS (Long Time Support) son más seguras, estables y, por lo tanto, más confiables. Además, tienen actualizaciones de software y soporte estándar por hasta cinco años.

En los siguientes capítulos de este anexo se detallará una guía de instalación de todas las herramientas empleadas durante la realización de este TFG, que son las necesarias para poder ejecutar un pipeline completo de Jenkins que nos permita alcanzar un DSOMM Level 1. Primero se deben instalar las herramientas de soporte como Java, Docker y Jenkins y después el resto de herramientas. Para cada herramienta normalmente hay varias maneras de instalarla, se debe acudir a su página web y ver cual es la que más nos conviene. Para la realización de este TFG se ha seleccionado la manera más sencilla de instalación, con la finalidad de que cualquier persona que lea este TFG, pueda instalar todas las herramientas y ejecutar el pipeline de Jenkins, sin que tenga que tener unos grandes conocimientos técnicos.

9.1.1 Instalación de Java

Es necesaria la instalación de Java para poder ejecutar Docker y Jenkins porque ambas son aplicaciones desarrolladas en lenguaje Java. Para cada versión específica de Docker y de Jenkins se necesitará una versión en concreto de Java. Por ejemplo, para Jenkins se puede revisar en la siguiente página web: <https://www.jenkins.io/doc/administration/requirements/java/>

Para Docker vale la misma versión de Java, por tanto, instalaremos la versión “OpenJDK JRE 11 de 64 bits”.

Primero actualizamos el índice de paquetes:

```
alberto@DevSecOpsTFG:/$ sudo apt update
[sudo] password for alberto:
Hit:1 http://es.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:3 http://es.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:4 http://es.archive.ubuntu.com/ubuntu jammy-backports InRelease [108 kB]
```

Ahora instalamos Java Runtime Environment (JRE) predeterminado:

```
alberto@DevSecOpsTFG:/$ sudo apt install default-jre
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  ca-certificates-java default-jre-headless fonts-dejavu-extra java-common
  libatk-wrapper-java libatk-wrapper-java-jni openjdk-11-jre
  openjdk-11-jre-headless
Suggested packages:
  fonts-ipafont-gothic fonts-ipafont-mincho fonts-wqy-microhei
  | fonts-wqy-zenhei
The following NEW packages will be installed:
  ca-certificates-java default-jre default-jre-headless fonts-dejavu-extra
  java-common libatk-wrapper-java libatk-wrapper-java-jni openjdk-11-jre
  openjdk-11-jre-headless
0 upgraded, 9 newly installed, 0 to remove and 5 not upgraded.
Need to get 44,8 MB of archives.
After this operation, 184 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

Verificamos que funciona:

```
alberto@DevSecOpsTFG:/$ java -version
openjdk version "11.0.19" 2023-04-18
OpenJDK Runtime Environment (build 11.0.19+7-post-Ubuntu-0ubuntu122.04.1)
OpenJDK 64-Bit Server VM (build 11.0.19+7-post-Ubuntu-0ubuntu122.04.1, mixed mode, sharing)
```

Instalamos también el JDK predeterminado por si necesitamos compilar algo:

```
alberto@DevSecOpsTFG:/$ sudo apt install default-jdk
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  default-jdk-headless libice-dev libpthread-stubs0-dev libsm-dev libx11-dev
  libxau-dev libxcb1-dev libxdmcp-dev libxt-dev openjdk-11-jdk
  openjdk-11-jdk-headless x11proto-dev xorg-sgml-doctools xtrans-dev
Suggested packages:
  libice-doc libsm-doc libx11-doc libxcb-doc libxt-doc openjdk-11-demo
  openjdk-11-source visualvm
The following NEW packages will be installed:
  default-jdk default-jdk-headless libice-dev libpthread-stubs0-dev libsm-dev
  libx11-dev libxau-dev libxcb1-dev libxdmcp-dev libxt-dev openjdk-11-jdk
  openjdk-11-jdk-headless x11proto-dev xorg-sgml-doctools xtrans-dev
0 upgraded, 15 newly installed, 0 to remove and 5 not upgraded.
Need to get 76,8 MB of archives.
After this operation, 90,5 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Verificamos que funciona:

```
alberto@DevSecOpsTFG:/$ javac -version
javac 11.0.19
```

9.1.2 Instalación de Docker

Primero actualizamos el repositorio de Ubuntu con “*sudo apt update*”:

```
alberto@DevSecOpsTFG:~$ sudo apt update
[sudo] password for alberto:
Hit:1 http://es.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:3 http://es.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:4 http://es.archive.ubuntu.com/ubuntu jammy-backports InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
5 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

Instalamos los paquetes necesarios con “*sudo apt install ca-certificates curl gnupg lsb-release*”:

```
alberto@DevSecOpsTFG:~$ sudo apt install ca-certificates curl gnupg lsb-release
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
lsb-release is already the newest version (11.1.0ubuntu4).
lsb-release set to manually installed.
ca-certificates is already the newest version (20230311ubuntu0.22.04.1).
ca-certificates set to manually installed.
gnupg is already the newest version (2.2.27-3ubuntu2.1).
gnupg set to manually installed.
The following NEW packages will be installed:
  curl
0 upgraded, 1 newly installed, 0 to remove and 5 not upgraded.
Need to get 194 kB of archives.
After this operation, 454 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

Añadimos la clave GPG oficial de Docker con los siguientes comandos:

```
alberto@DevSecOpsTFG:~$ sudo install -m 0755 -d /etc/apt/keyrings
alberto@DevSecOpsTFG:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg
| sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
alberto@DevSecOpsTFG:~$ sudo chmod a+r /etc/apt/keyrings/docker.gpg
```

Configuramos el repositorio estable con “echo “deb [arch=\$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu “\$(. /etc/os-release && echo “\$VERSION_CODENAME”)” stable” | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null”:

```
alberto@DevSecOpsTFG:~$ echo "deb [arch=$(dpkg --print-architecture)] signed-by
=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu “$(. /et
c/os-release && echo “$VERSION_CODENAME”)” stable" | sudo tee /etc/apt/sources.l
ist.d/docker.list > /dev/null
```

Listamos el directorio con “ls -l /etc/apt/sources.list.d/” y comprobamos que el nuevo repositorio estable de docker se ha añadido:

```
alberto@DevSecOpsTFG:~$ ls -l /etc/apt/sources.list.d/
total 4
-rw-r--r-- 1 root root 110 mai  26 19:40 docker.list
```

Actualizamos los repositorios con “sudo apt update”, donde ya veremos que coge del repositorio de Docker:

```
alberto@DevSecOpsTFG:~$ sudo apt update
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:2 https://download.docker.com/linux/ubuntu jammy InRelease [48,9 kB]
Hit:3 http://es.archive.ubuntu.com/ubuntu jammy InRelease
Get:4 http://es.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:5 https://download.docker.com/linux/ubuntu jammy/stable amd64 Packages [19,2
kB]
Get:6 http://es.archive.ubuntu.com/ubuntu jammy-backports InRelease [108 kB]
Get:7 http://es.archive.ubuntu.com/ubuntu jammy-updates/main i386 Packages [398
kB]
Get:8 http://es.archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [618
kB]
Get:9 http://es.archive.ubuntu.com/ubuntu jammy-updates/universe i386 Packages [
614 kB]
Get:10 http://es.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages
[902 kB]
Fetched 2 937 kB in 3s (1 159 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
5 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

Instalamos Docker CE, con “sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin”:

```
alberto@DevSecOpsTFG:~$ sudo apt install docker-ce docker-ce-cli containerd.io d
ocker-buildx-plugin docker-compose-plugin
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  docker-ce-rootless-extras libslirp0 pigz slirp4netns
Suggested packages:
  aufs-tools cgroupfs-mount | cgroup-lite
The following NEW packages will be installed:
  containerd.io docker-buildx-plugin docker-ce docker-ce-cli
  docker-ce-rootless-extras docker-compose-plugin libslirp0 pigz slirp4netns
0 upgraded, 9 newly installed, 0 to remove and 5 not upgraded.
Need to get 111 MB of archives.
After this operation, 401 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

Comprobamos la versión que nos ha instalado de Docker con “sudo docker -v”:

```
alberto@DevSecOpsTFG:~$ sudo docker -v
Docker version 24.0.2, build cb74dfc
```

Descargamos y lanzamos un contenedor de pruebas con “*sudo docker run hello-world*”:

```
alberto@DevSecOpsTFG:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
719385e32844: Pull complete
Digest: sha256:fc6cf906cbfa013e80938cdf0bb199fbd86d6e3e013783e5a766f50f5dbce0
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Comprobamos el estado del contenedor con “*sudo docker ps -a*”:

```
alberto@DevSecOpsTFG:~$ sudo docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS          NAMES
0bf3eeb041ad  hello-world   "/hello"                 About a minute ago  Exited (0)     About a minute ago  hardcore_shaw
```

9.1.3 Instalación de Docker Compose

Docker simplifica la administración de procesos de aplicaciones en contenedores, que son más ligeros y sencillos en recursos en comparación con las máquinas virtuales. Esto permite a los desarrolladores dividir un entorno de aplicación en servicios aislados. Para las aplicaciones que dependen de múltiples servicios, organizar los contenedores para iniciar, comunicar y apagar juntos puede volverse complicado.

Docker Compose es una herramienta que facilita la ejecución de entornos de aplicaciones multi contenedor utilizando definiciones establecidas en un archivo YAML. Con las definiciones de servicio, se pueden crear entornos personalizables con múltiples contenedores que pueden compartir redes y volúmenes de datos.

Para instalar Docker Compose primero actualizamos el índice de paquetes:

```
alberto@DevSecOpsTFG:~/TFG_DevSecOps$ sudo apt update
[sudo] contraseña para alberto:
Obj:1 http://es.archive.ubuntu.com/ubuntu jammy InRelease
Obj:2 http://es.archive.ubuntu.com/ubuntu jammy-updates InRelease
Obj:3 http://security.ubuntu.com/ubuntu jammy-security InRelease
Ign:4 https://pkg.jenkins.io/debian-stable binary/ InRelease
Obj:5 http://es.archive.ubuntu.com/ubuntu jammy-backports InRelease
Obj:6 https://pkg.jenkins.io/debian-stable binary/ Release
Des:7 https://pkg.jenkins.io/debian-stable binary/ Release.gpg [833 B]
Obj:8 https://download.docker.com/linux/ubuntu jammy InRelease
Descargados 833 B en 1s (558 B/s)
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se pueden actualizar 5 paquetes. Ejecute «apt list --upgradable» para verlos.
```

Ahora descargamos e instalamos la última versión disponible de Docker Compose con el comando “*sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-\$(uname -s)-\$(uname -m)" -o /usr/local/bin/docker-compose*”:

```
alberto@DevSecOpsTFG:~/TFG_DevSecOps$ sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload  Total   Spent    Left     Speed
  0     0    0     0    0     0     0     0  0:00:00  0:00:00  0:00:00     0
100 52.0M 100 52.0M    0     0 18.3M    0  0:00:02  0:00:02  0:00:00 36.7M
```


Ahora solo queda aplicar los permisos de ejecución sobre el binario descargado, para ello utilizamos el siguiente comando “`sudo chmod +x /usr/local/bin/docker-compose`”:

```
alberto@DevSecOpsTFG:~/TFG_DevSecOps$ sudo chmod +x /usr/local/bin/docker-compose
alberto@DevSecOpsTFG:~/TFG_DevSecOps$
```

Para comprobar que está correctamente instalado utilizamos “`docker-compose --version`”:

```
alberto@DevSecOpsTFG:~/TFG_DevSecOps$ docker-compose --version
Docker Compose version v2.18.1
```

9.1.4 Instalación de Jenkins

Para instalar Jenkins primero actualizamos el repositorio de Ubuntu con “`sudo apt update`”:

```
alberto@DevSecOpsTFG:~$ sudo apt update
[sudo] contraseña para alberto:
Obj:1 https://download.docker.com/linux/ubuntu jammy InRelease
Obj:2 http://es.archive.ubuntu.com/ubuntu jammy InRelease
Obj:3 http://security.ubuntu.com/ubuntu jammy-security InRelease
Obj:4 http://es.archive.ubuntu.com/ubuntu jammy-updates InRelease
Des:5 http://es.archive.ubuntu.com/ubuntu jammy-backports InRelease [108 kB]
Descargados 108 kB en 2s (59,0 kB/s)
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se pueden actualizar 5 paquetes. Ejecute «apt list --upgradable» para verlos.
```

Añadimos la clave del repositorio con “`curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null`”:

```
alberto@DevSecOpsTFG:~$ curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \
/usr/share/keyrings/jenkins-keyring.asc > /dev/null
```

Añadimos los repositorios y actualizamos con “`echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null`”:

```
alberto@DevSecOpsTFG:~$ echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null
```

Actualizamos con un “`sudo apt update`”:

```
alberto@DevSecOpsTFG:~$ sudo apt update
Obj:1 http://es.archive.ubuntu.com/ubuntu jammy InRelease
Des:2 http://es.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Des:3 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Des:4 http://es.archive.ubuntu.com/ubuntu jammy-backports InRelease [108 kB]
Ign:5 https://pkg.jenkins.io/debian-stable binary/ InRelease
Obj:6 https://download.docker.com/linux/ubuntu jammy InRelease
Des:7 https://pkg.jenkins.io/debian-stable binary/ Release [2.044 B]
Des:8 https://pkg.jenkins.io/debian-stable binary/ Release.gpg [833 B]
Des:9 http://es.archive.ubuntu.com/ubuntu jammy-updates/main i386 Packages [398 kB]
Des:10 http://es.archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [618 kB]
Des:11 http://es.archive.ubuntu.com/ubuntu jammy-updates/main amd64 c-n-f Metadata [14,6 kB]
Des:12 http://es.archive.ubuntu.com/ubuntu jammy-updates/universe i386 Packages [614 kB]
Des:13 http://es.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [902 kB]
Des:14 https://pkg.jenkins.io/debian-stable binary/ Packages [24,8 kB]
Descargados 2.912 kB en 2s (1.247 kB/s)
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se pueden actualizar 5 paquetes. Ejecute «apt list --upgradable» para verlos.
```

Instalamos el paquete desde APT con “`sudo apt install jenkins`”:


```
alberto@DevSecOpsTFG:~$ sudo apt install jenkins
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
 net-tools
Se instalarán los siguientes paquetes NUEVOS:
 jenkins net-tools
0 actualizados, 2 nuevos se instalarán, 0 para eliminar y 5 no actualizados.
Se necesita descargar 98,1 MB de archivos.
Se utilizarán 99,3 MB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] s
```

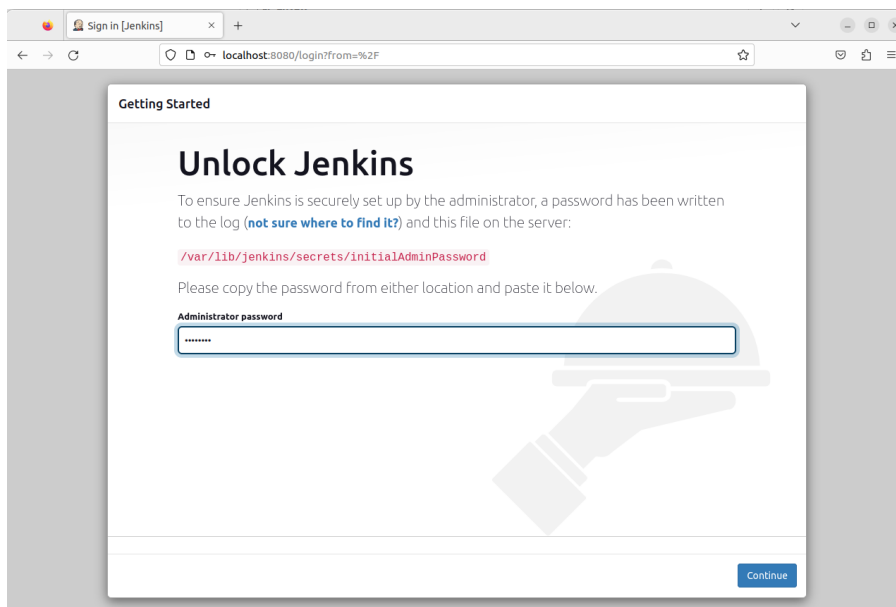
Una vez instalado, iniciamos el servicio y lo habilitamos al arranque del servidor con “`systemctl start jenkins.service`” y “`systemctl enable jenkins.service`”, nos pedirá la password de root si no somos administradores:

```
alberto@DevSecOpsTFG:~$ systemctl start jenkins.service
alberto@DevSecOpsTFG:~$ systemctl start jenkins.service
```

Revisamos el estado del servicio con “`systemctl status jenkins.service`” y vemos que está funcionando:

```
alberto@DevSecOpsTFG:~$ systemctl status jenkins.service
● jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2023-05-26 22:15:54 CEST; 3min 20s ago
     Main PID: 7825 (java)
       Tasks: 42 (limit: 4614)
      Memory: 1.1G
             CPU: 1min 41.904s
      CGroup: /system.slice/jenkins.service
              └─7825 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/
lines 1-9/9 (END)
```

Ya tenemos instalado Jenkins, ahora debemos configurarlo. Para ellos accedemos mediante un navegador web utilizando la IP pública del servidor y el puerto 8080. La primera vez que accedemos nos pide cambiar la licencia de administrador:



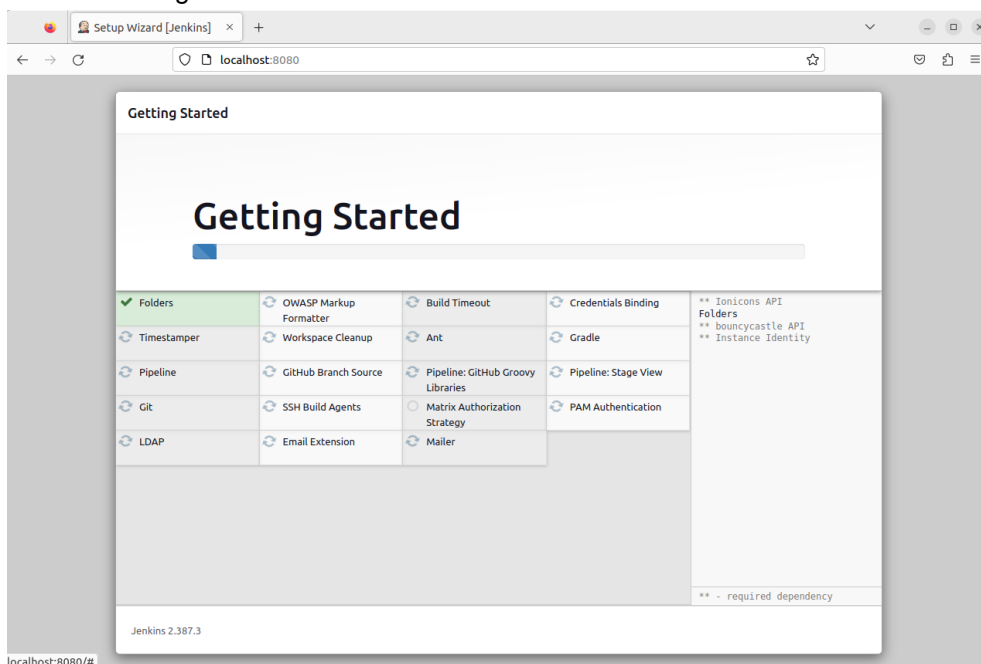
Debemos meter como contraseña inicial la del contenido del fichero que nos indican, para ellos poneemos en un terminal “`sudo cat /var/lib/Jenkins/initialAdminPassword`”:

```
alberto@DevSecOpsTFG:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
14d7aa812c0a438faed2a541b82159ca
```

La siguiente pantalla nos indicará si instalar los plugins por defecto o seleccionar los plugins manualmente, lo más recomendable es marcar la primera opción (suggested plugins):



Los irá descargando e instalando automáticamente:



Una vez finalice la instalación, deberemos de crear la cuenta administradora (la dirección de email no es necesario que exista):

Getting Started

Create First Admin User

Usuario

Contraseña

Confirma la contraseña

Nombre completo

Dirección de email

Jenkins 2.387.3 Skip and continue as admin [Save and Continue](#)

A continuación, podemos seleccionar la URL de acceso (podemos dejar la IP de nuestro servidor o configurar un Nginx Proxy para acceder via dominio):

Getting Started

Instance Configuration

Jenkins URL:

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD_URL environment variable provided to build steps.

The proposed default value shown is not saved yet and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Jenkins 2.387.3 Not now [Save and Finish](#)

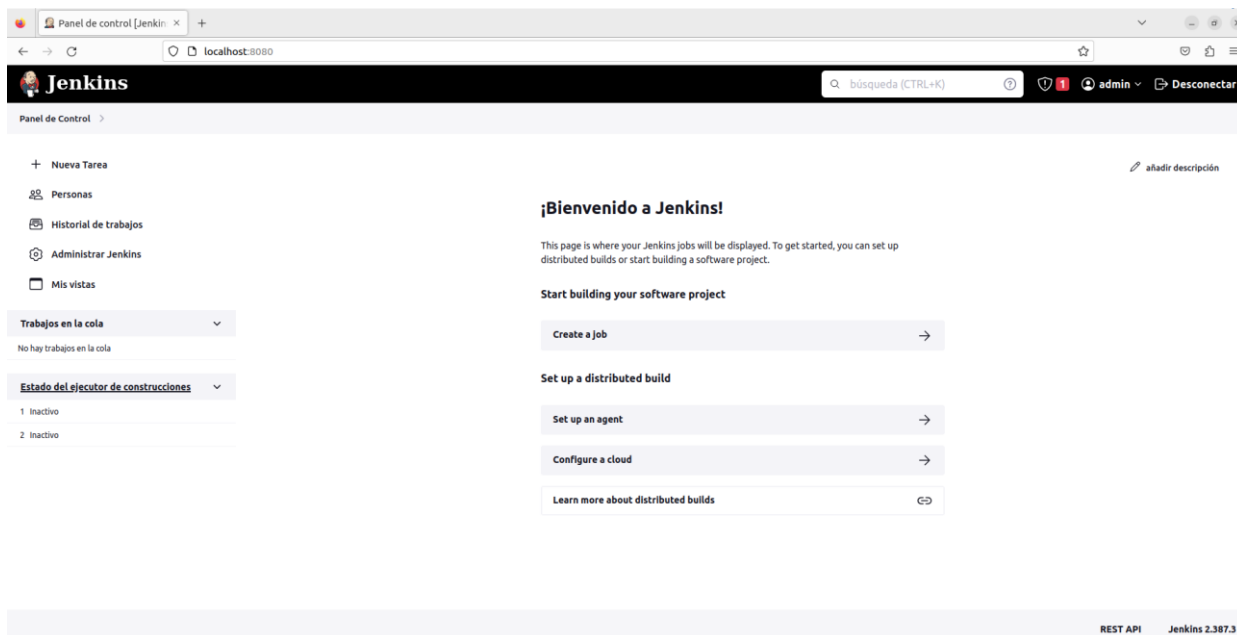
Ahora ya podremos acceder a Jenkins:

Getting Started

Jenkins is ready!

Your Jenkins setup is complete.

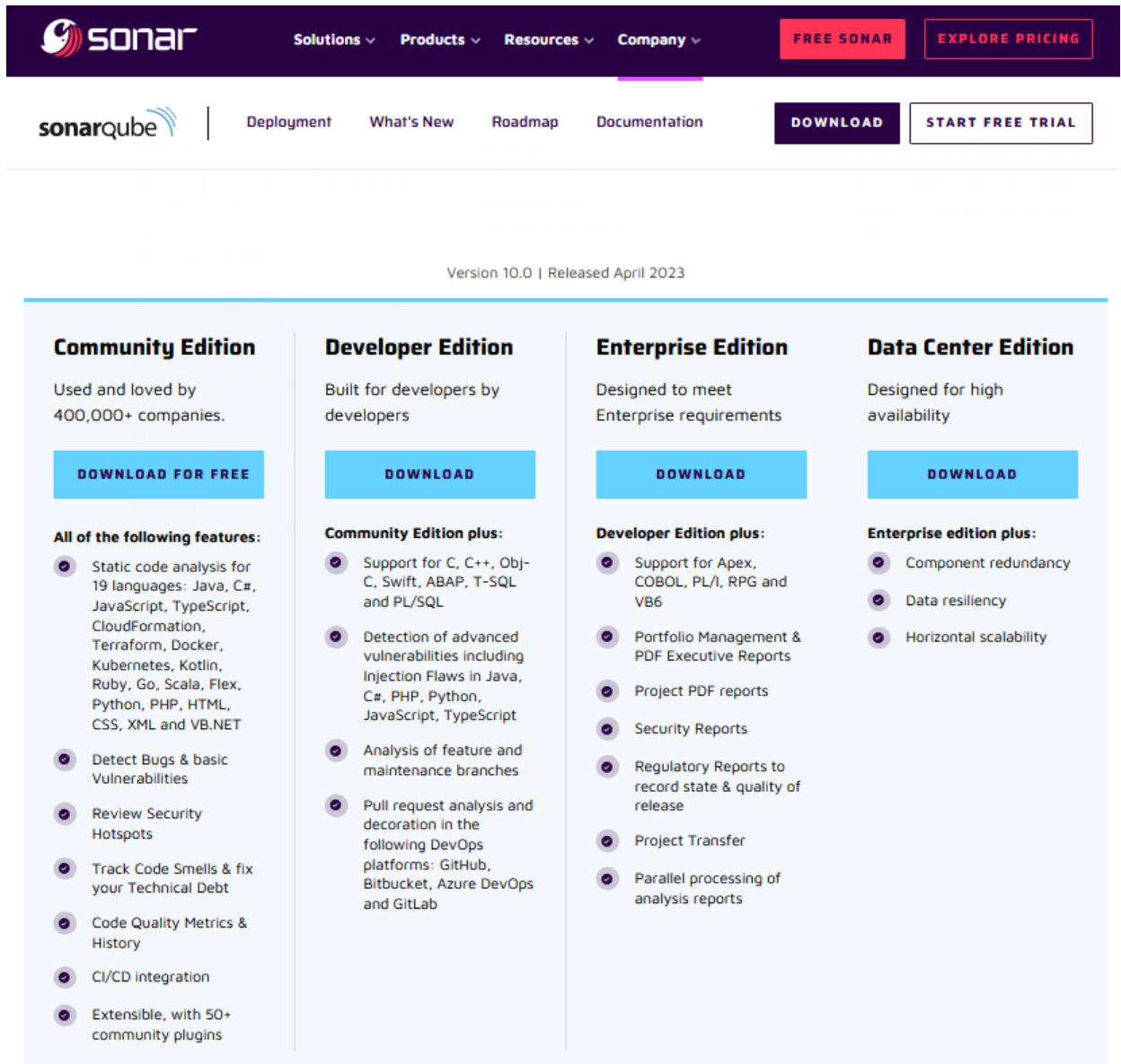
[Start using Jenkins](#)



9.1.5 Instalación de SonarQube

SonarQube es una plataforma de código abierto para la inspección continua de la calidad del código a través de diferentes herramientas de análisis estático de código fuente. Proporciona métricas que ayudan a mejorar la calidad del código al permitir que los equipos de desarrollo rastreen e identifiquen errores y vulnerabilidades de seguridad para mantener el código limpio.

SonarQube tiene varias ediciones, la Community que es la Open Source y otras tres ediciones de pago. La diferencia entre las distintas ediciones son el número de lenguajes que soportan y las funcionalidades que incluyen. Las características de las diferentes ediciones se pueden visualizar en la siguiente imagen de la página del proyecto [31]:



Version 10.0 | Released April 2023

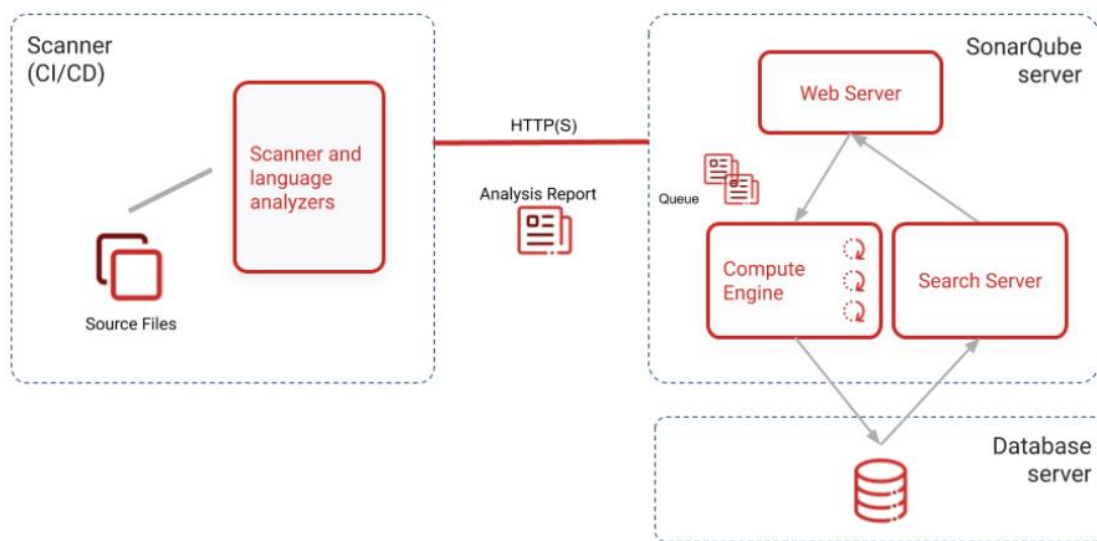
Community Edition	Developer Edition	Enterprise Edition	Data Center Edition
Used and loved by 400,000+ companies.	Built for developers by developers	Designed to meet Enterprise requirements	Designed for high availability
DOWNLOAD FOR FREE	DOWNLOAD	DOWNLOAD	DOWNLOAD
All of the following features: <ul style="list-style-type: none"> Static code analysis for 19 languages: Java, C#, JavaScript, TypeScript, CloudFormation, Terraform, Docker, Kubernetes, Kotlin, Ruby, Go, Scala, Flex, Python, PHP, HTML, CSS, XML and VB.NET Detect Bugs & basic Vulnerabilities Review Security Hotspots Track Code Smells & fix your Technical Debt Code Quality Metrics & History CI/CD integration Extensible, with 50+ community plugins 	Community Edition plus: <ul style="list-style-type: none"> Support for C, C++, Obj-C, Swift, ABAP, T-SQL and PL/SQL Detection of advanced vulnerabilities including Injection Flaws in Java, C#, PHP, Python, JavaScript, TypeScript Analysis of feature and maintenance branches Pull request analysis and decoration in the following DevOps platforms: GitHub, Bitbucket, Azure DevOps and GitLab 	Developer Edition plus: <ul style="list-style-type: none"> Support for Apex, COBOL, PL/I, RPG and VB6 Portfolio Management & PDF Executive Reports Project PDF reports Security Reports Regulatory Reports to record state & quality of release Project Transfer Parallel processing of analysis reports 	Enterprise edition plus: <ul style="list-style-type: none"> Component redundancy Data resiliency Horizontal scalability

Para la realización de este TFG vamos a utilizar la versión Community, que no tiene escáner de seguridad, pero que se puede conseguir descargando la herramienta SonarScanner de su página web [32].

Esta herramienta viene empaquetada como un fichero zip, que lo copiamos dentro de nuestro ordenador a la carpeta que queramos, lo descomprimos y lo añadimos al PATH, para poder ejecutarlo desde cualquier sitio con el siguiente comando:

```
alberto@DevSecOpsTFG:~/TFG_DevSecOps/SAST/SonarQube/sonar-scanner/bin$ export PATH=$PATH:~/TFG_DevSecOps/SAST/SonarQube/sonar-scanner/bin
alberto@DevSecOpsTFG:~/TFG_DevSecOps/SAST/SonarQube/sonar-scanner/bin$
```

Para instalar SonarQube, se puede instalar localmente en el sistema, pero además necesita instalar una base de datos como PostgreSQL. La arquitectura es la siguiente [33]:



Para simplificar ese proceso también es posible descargar las imágenes Docker de SonarQube y de PostgreSQL desde el repositorio oficial de Docker Hub [34] y gestionarlas desde Docker Compose utilizando un fichero Docker-compose.yml como el siguiente:

```
version: '2'

services:
  sonarqube:
    image: sonarqube
    ports:
      - "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:
    driver: bridge

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

Para ejecutarlo y que funciones las dos imágenes, la de SonarQube y la de PostgreSQL, abrimos una terminal de LINUX y primero ejecutamos el siguiente comando “*sudo sysctl -w vm.max_map_count=262144*” para evitar que Elasticsearch se quede sin memoria y nos de problemas:

```
alberto@DevSecOpsTFG:~/TFG_DevSecOps/SAST/SonarQube$ sudo sysctl -w vm.max_map_count=262144
[sudo] contraseña para alberto:
vm.max_map_count = 262144
```

Después ejecutamos el siguiente comando para cargar el fichero yml con Docker Compose “*sudo docker-compose -f sonarqube-compose.yml up*”:


```

alberto@DevSecOpsTFG:~/TFG_DevSecOps/SAST/SonarQube$ sudo docker-compose -f sonarqube-compose.yml up
[+] Running 21/21
 ✓ db 13 layers [██████████] 0B/0B Pulled 42.1s
 ✓ f03b40093957 Pull complete 13.7s
 ✓ 9d674c93414d Pull complete 14.7s
 ✓ de781e8e259a Pull complete 15.6s
 ✓ 5ea6efaf51f6 Pull complete 16.2s
 ✓ b078d5f4ac82 Pull complete 18.5s
 ✓ 97f84fb2a918 Pull complete 19.2s
 ✓ 5a6bf2f43fb8 Pull complete 19.3s
 ✓ f1a40e88Fea4 Pull complete 19.3s
 ✓ 4be673794a1a Pull complete 36.6s
 ✓ 9d72f84fb861 Pull complete 36.7s
 ✓ 5d52569da92e Pull complete 38.3s
 ✓ 5d48fbe991ff Pull complete 39.8s
 ✓ 4ae692d11ad3 Pull complete 40.1s
 ✓ sonarqube 6 layers [██████████] 0B/0B Pulled 42.3s
 ✓ 1bc677758ad7 Pull complete 15.8s
 ✓ 0d0e0ecb256a Pull complete 22.2s
 ✓ 212512b6dedf Pull complete 27.5s
 ✓ 648d9d544695 Pull complete 27.6s

```

Tarda un poco en descargar todo e instalarlo, pero al final nos dice que todo está operativo:

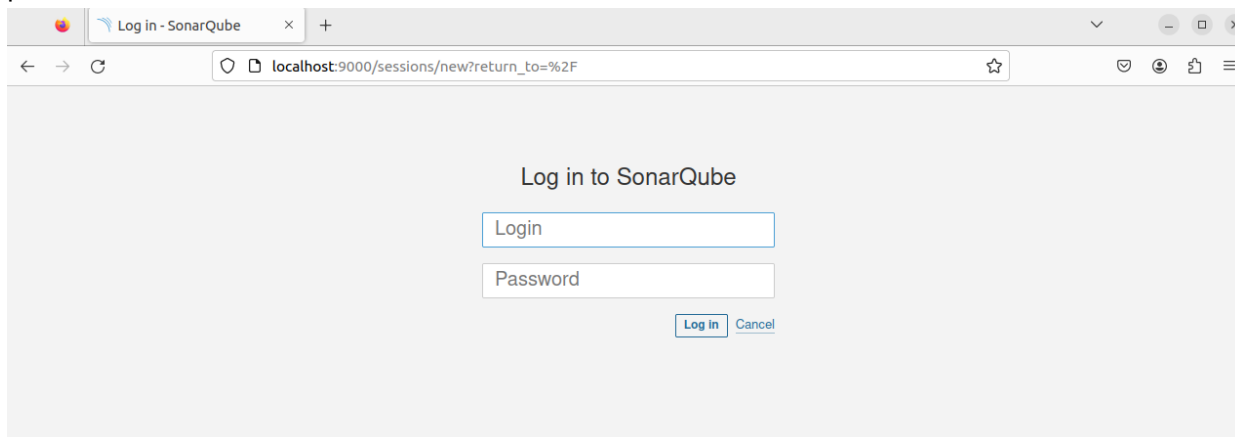
```

sonarqube-sonarqube-1 | 2023.05.27 11:58:32 WARN app[][startup] #####
#####
sonarqube-sonarqube-1 | 2023.05.27 11:58:32 INFO web[][o.s.s.p.Platform] Web Server is operational
sonarqube-sonarqube-1 | 2023.05.27 11:58:34 INFO ce[][o.sonar.db.Database] Create JDBC data source fo
r jdbc:h2:tcp://127.0.0.1:9092/sonar;NON_KEYWORDS=VALUE
sonarqube-sonarqube-1 | 2023.05.27 11:58:34 INFO ce[][c.z.h.HikariDataSource] HikariPool-1 - Starting
...
sonarqube-sonarqube-1 | 2023.05.27 11:58:34 INFO ce[][c.z.h.p.HikariPool] HikariPool-1 - Added connec
tion conn0: url=jdbc:h2:tcp://127.0.0.1:9092/sonar
sonarqube-sonarqube-1 | 2023.05.27 11:58:34 INFO ce[][c.z.h.HikariDataSource] HikariPool-1 - Start co
mpleted.
sonarqube-sonarqube-1 | 2023.05.27 11:58:34 WARN ce[][o.s.db.dialect.H2] H2 database should be used f
or evaluation purpose only.
sonarqube-sonarqube-1 | 2023.05.27 11:58:39 INFO ce[][o.s.s.p.ServerFileSystemImpl] SonarQube home: /
opt/sonarqube
sonarqube-sonarqube-1 | 2023.05.27 11:58:40 INFO ce[][o.s.c.c.CePluginRepository] Load plugins
sonarqube-sonarqube-1 | 2023.05.27 11:58:51 INFO ce[][o.s.c.c.ComputeEngineContainerImpl] Running Com
munity edition
sonarqube-sonarqube-1 | 2023.05.27 11:58:51 INFO ce[][o.s.ce.app.CeServer] Compute Engine is started
sonarqube-sonarqube-1 | 2023.05.27 11:58:51 INFO app[][o.s.a.SchedulerImpl] Process[ce] is up
sonarqube-sonarqube-1 | 2023.05.27 11:58:51 INFO app[][o.s.a.SchedulerImpl] SonarQube is operational

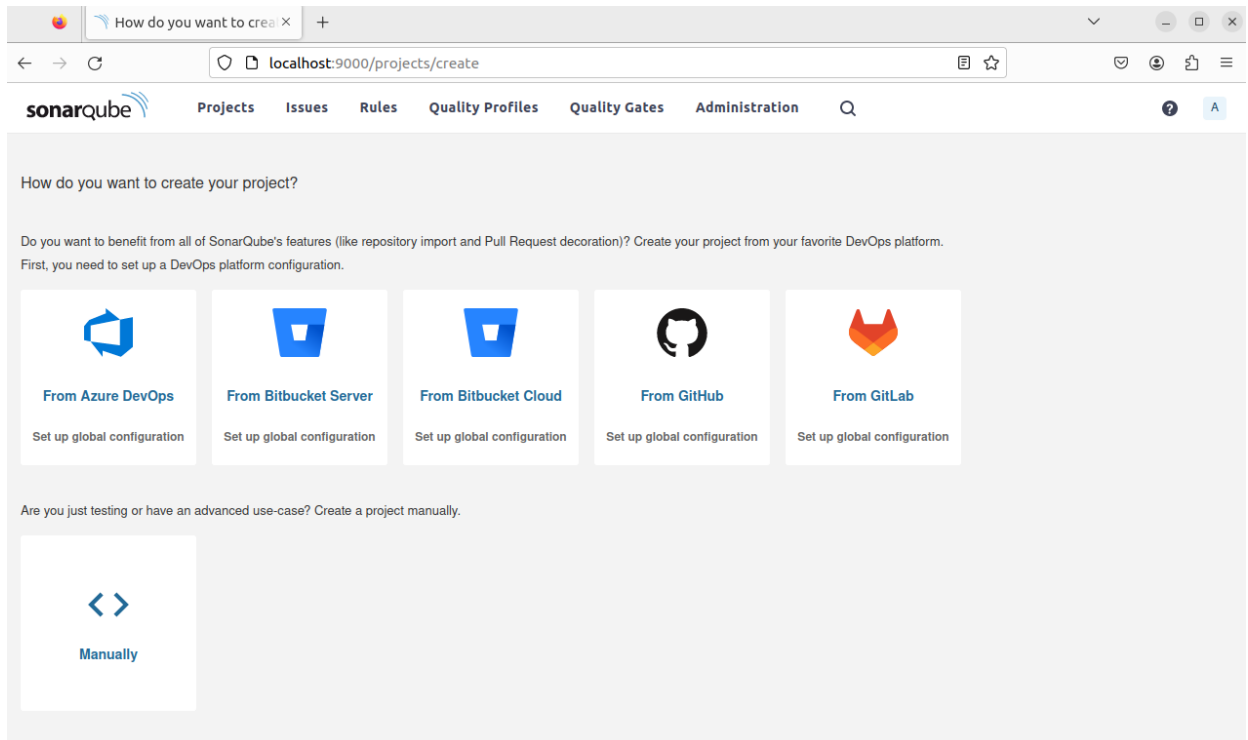
```

Es muy importante no cerrar esa terminal porque se caería el servidor y nos quedaríamos sin SonarQube.

Una vez arrancados los contenedores, abrimos un navegador web y accedemos a <http://localhost:9000> para acceder a la consola de SonarQube:



El usuario y la contraseña predeterminados son admin y admin respectivamente. La primera vez que accedamos nos pedirá que cambiemos la contraseña. Después de cambiar la contraseña ya entraremos en la página de SonarQube donde podremos crear nuevos proyectos con las opciones que nos muestra:



9.1.6 Instalación de Retire.JS

Primero actualizamos el repositorio de Ubuntu:

```
alberto@DevSecOpsTFG:~/Aplicaciones/SAST$ sudo apt update
Hit:1 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:2 http://es.archive.ubuntu.com/ubuntu jammy InRelease
Hit:3 http://es.archive.ubuntu.com/ubuntu jammy-updates InRelease
Get:4 http://es.archive.ubuntu.com/ubuntu jammy-backports InRelease [108 kB]
Fetched 108 kB in 1s (84,9 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
5 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

Instalamos el paquete nodejs:

```
alberto@DevSecOpsTFG:~/Aplicaciones/SAST$ sudo apt install nodejs
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  javascript-common libc-ares2 libjs-highlight.js libnode72 nodejs-doc
Suggested packages:
  apache2 | lighttpd | httpd npm
The following NEW packages will be installed:
  javascript-common libc-ares2 libjs-highlight.js libnode72 nodejs nodejs-doc
0 upgraded, 6 newly installed, 0 to remove and 5 not upgraded.
Need to get 13,7 MB/13,7 MB of archives.
After this operation, 53,9 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

Instalamos el gestor de paquetes npm:

```
alberto@DevSecOpsTFG:~/Aplicaciones/SAST$ sudo apt install npm
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
```

Instalamos la aplicación Retire:

```
alberto@DevSecOpsTFG:~/Aplicaciones/SAST$ sudo npm install -g retire
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'retire@4.2.1',
npm WARN EBADENGINE   required: { node: '>= 14.0.0' },
npm WARN EBADENGINE   current: { node: 'v12.22.9', npm: '8.5.1' }
npm WARN EBADENGINE }
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'commander@10.0.1',
npm WARN EBADENGINE   required: { node: '>=14' },
npm WARN EBADENGINE   current: { node: 'v12.22.9', npm: '8.5.1' }
npm WARN EBADENGINE }
added 66 packages, and audited 67 packages in 10s

1 package is looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Podemos verificar que se ha instalado ejecutándola por línea de comandos:

```
alberto@DevSecOpsTFG:~/Aplicaciones/SAST$ retire
retire.js v4.2.1
Downloading https://raw.githubusercontent.com/RetireJS/retire.js/master/repository/jsrepository.json ...
```

9.1.7 Instalación de OWASP Dependency-Check

OWASP Dependency-Check es una herramienta que nos permite identificar las dependencias de nuestro proyecto y comprobar si hay alguna de ellas que tiene vulnerabilidades conocidas.

Para instalarlo, hay que descargarlo de la página web del proyecto [35]. Se descarga en formato zip y simplemente debemos descomprimirlo en la carpeta donde deseemos que se ejecute. Después para ejecutarlo tan solo debemos pasarle el camino raíz donde se encuentre el código a analizar y el nombre del fichero que queremos que nos genere con el reporte. Por ejemplo, para saber la versión que tenemos instalada ejecutaríamos el comando `./dependency-check.sh -v`:

```
alberto@DevSecOpsTFG:~/TFG_DevSecOps/SAST/dependency-check/bin$ ./dependency-check.sh -v
Dependency-Check Core version 8.2.1
```

Si queremos ejecutarlo desde cualquier carpeta del sistema podemos crear un link simbólico utilizando el siguiente comando `ln -s OWASP_path_completo_instalacion/dependency-check.sh /usr/bin/dependency-check.sh`.

9.1.8 Instalación de NodeJSScan

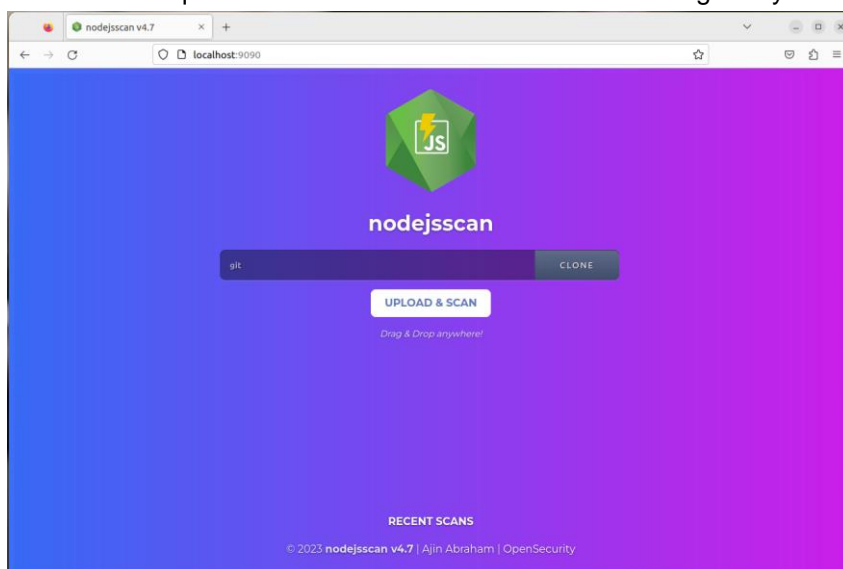
Para instalarlo se debe descargar su imagen de docker con `sudo docker pull opensecurity/nodejsscan:latest`:

```
alberto@DevSecOpsTFG:~$ sudo docker pull opensecurity/nodejsscan:latest
[sudo] contraseña para alberto:
latest: Pulling from opensecurity/nodejsscan
e9995326b091: Pull complete
a0cb03f17886: Pull complete
bb26f7e78134: Pull complete
c8e073b7ae91: Pull complete
99b5b1679915: Pull complete
55c520fc03c5: Pull complete
d0ac84d6672c: Pull complete
4effb95d5849: Pull complete
0d6d87c178f1: Pull complete
43c8b5f8bcaa: Pull complete
825548df579f: Pull complete
3878b693881a: Pull complete
c0e50aa09815: Pull complete
3e292d6ef8ed: Pull complete
1e70e3657e3a: Pull complete
8b4173f4eec1: Pull complete
00f45a23d3b5: Pull complete
3a775f9d06c6: Pull complete
Digest: sha256:e7968e12e8343365203443e6efec3c8587bf2a3ba6b37d5f803f2d4b7ef8d289
Status: Downloaded newer image for opensecurity/nodejsscan:latest
docker.io/opensecurity/nodejsscan:latest
```

Luego se debe levantar el contenedor con "`sudo docker run --rm -d -it -p 9090:9090 opensecurity/nodejsscan:latest`":

```
alberto@DevSecOpsTFG:~$ sudo docker run --rm -d -it -p 9090:9090 opensecurity/no
dejsscan:latest
f12bbf8de8dcf6303ffa6e6898d282da088e8d4c036868a0bc79c8f98d90b3be
```

Para verificar que está en funcionamiento abrimos el navegador y vamos a "`localhost:9090`":



9.1.9 Instalación de Nuclei

Lo descargamos desde su página web [36] como un fichero zip, que descomprimos en una carpeta de nuestra máquina.

Debemos darle permisos de ejecución con el comando "`chmod +x nuclei`". Después lo ejecutamos en una terminal del sistema operativo poniendo solo "`nuclei`". La primera vez que la ejecutemos se conectará a Internet y se descargará todas las plantillas (en el momento de la realización de este TFG son 5958):


```
alberto@DevSecOpsTFG:~/TFG_DevSecOps/DAST/Wapiti/wapiti3-3.1.7/bin$ ./wapiti --list-modules

Wapiti 3.1.7 (wapiti-scanner.github.io)
[*] Available modules:
  backup
    Uncover backup files on the web server.

  brute_login_form
    Attempt to login on authentication forms using known weak credentials (like admin/admin
  ).

  buster
    Brute force paths on the web-server to discover hidden files and directories.

  cookieflags (used by default)
    Evaluate the security of cookies on the website.
```

9.1.11 Instalación de ZAP Proxy

Es el proyecto insignia de la comunidad OWASP. Es un proxy web, esto es captura las peticiones http y luego las respuestas y así analizarlas. Es un framework completo para pentesting manual y para tests de webs de forma automática. Está desarrollado en Java. Se puede descargar desde su página web [38], pulsando sobre el icono “Linux Installer”.

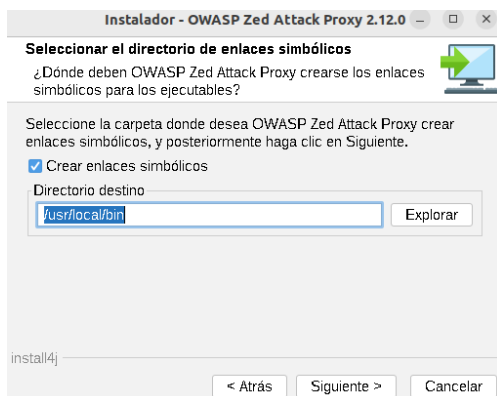
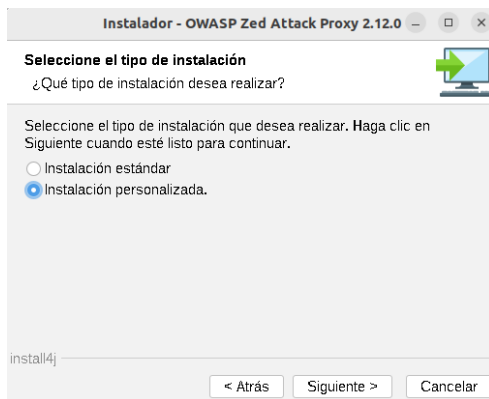
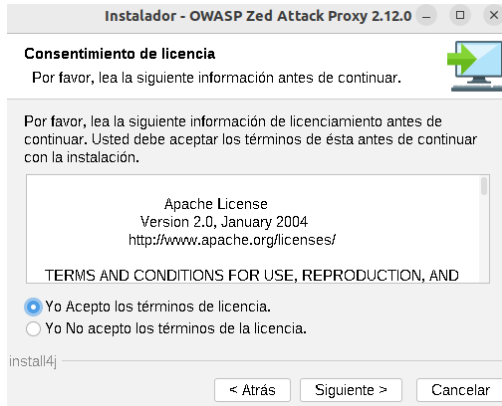
Para poder ejecutar el shell script debemos darle permisos de ejecución mediante el comando “*chmod +x ZAP_2_12_0_unix.sh*”:

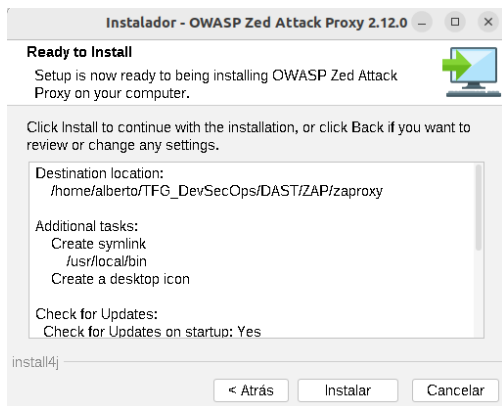
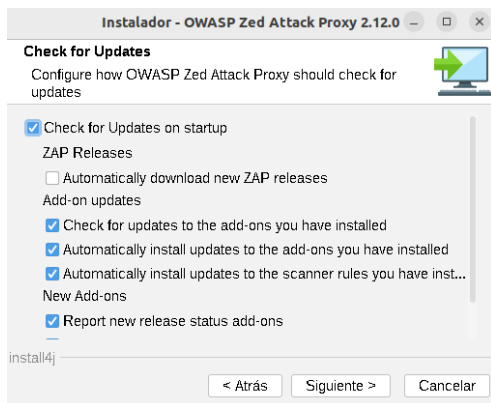
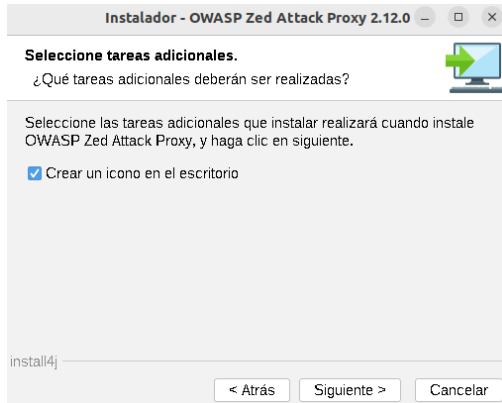
```
alberto@DevSecOpsTFG:~/TFG_DevSecOps/DAST/ZAP$ chmod +x ZAP_2_12_0_unix.sh
```

Después ejecutar el script “*sudo ./ZAP_2_12_0_unix.sh*”, seleccionando la “Custom installation” para poder indicarle el path donde instalarlo, porque sino lo instala en /opt/zaproxy:

```
alberto@DevSecOpsTFG:~/TFG_DevSecOps/DAST/ZAP$ sudo ./ZAP_2_12_0_unix.sh
[sudo] contraseña para alberto:
Starting Installer ...
Gtk-Message: 17:36:41.673: Failed to load module "canberra-gtk-module"
```

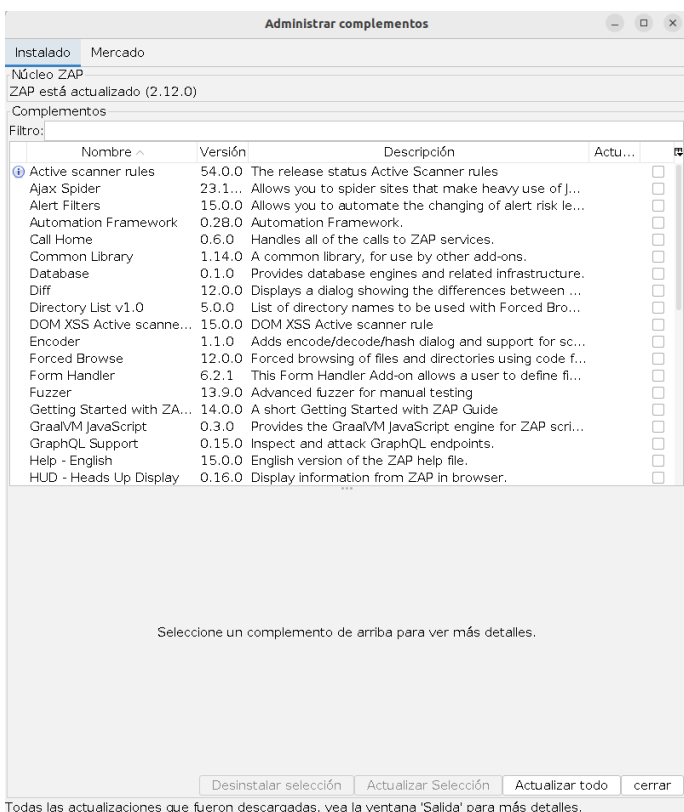






Una vez esté instalado, podemos ejecutarlo con el comando “`sudo ./zap.sh`” y al hacerlo por primera vez pinchar en “Actualizar todo”, esperar y reiniciar la herramienta:

```
alberto@DevSecOpsTFG: ~/TFG_DevSecOps/DAST/ZAP/zaproxy$ sudo ./zap.sh
Found Java version 11.0.19
Available memory: 3923 MB
Using JVM args: -Xmx980m
3243 [main] INFO org.parosproxy.paros.Constant - Copying default configuration to /root/.ZAP/config.xml
3867 [main] INFO org.parosproxy.paros.Constant - Creating directory /root/.ZAP/session
3867 [main] INFO org.parosproxy.paros.Constant - Creating directory /root/.ZAP/dirbuster
3868 [main] INFO org.parosproxy.paros.Constant - Creating directory /root/.ZAP/fuzzers
3868 [main] INFO org.parosproxy.paros.Constant - Creating directory /root/.ZAP/plugin
4269 [main] INFO org.zaproxy.zap.GuiBootstrap - OWASP ZAP 2.12.0 started 27/05/2023, 17:49:24 with home /root/.ZAP/
```



9.1.12 Instalación de Trivy

Trivy es una escaner de vulnerabilidades que es capaz de analizar imágenes de contenedores, sistemas de ficheros, repositorios Git y problemas en ficheros de configuración.

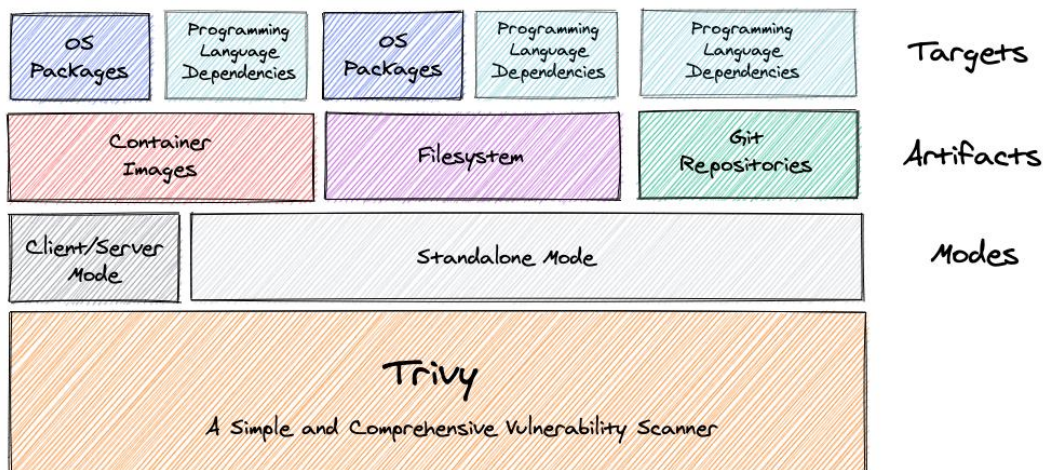


Figura 22. Trivy

Fuente: <https://aquasecurity.github.io/trivy/v0.17.2/>

Se puede descargar desde su página web y también se puede utilizar como una imagen de Docker.

Si se descarga como fichero comprimido (.gz), tan solo hay que descomprimirlo en una carpeta y ejecutarlo. Por ejemplo, para saber la versión ejecutamos `./trivy -v`:

```
alberto@DevSecOpsTFG:~/TFG_DevSecOps/Otras_Herramientas_Seguridad/Trivy/trivy_0.41.0_Linux-64bit$ ./trivy -v
Version: 0.41.0
```