



MÁSTER PRESENCIAL EN CIBERSEGURIDAD – 5ªEd.

DevSecOps - Cómo integrar la seguridad en procesos DevOps

TFM elaborado por: Jose María Acuña Morgado

Tutor de TFM: Javier Rueda Pérez

- Madrid, 9 de diciembre de 2020 -

Resumen

Las aplicaciones de *software* son complejas y pueden ser vulnerables a una amplia variedad de problemas de seguridad. La cultura empresarial frecuentemente sitúa la seguridad en la fase final del ciclo de vida del desarrollo de *software*.

DevSecOps se centra en desplazar la seguridad hacia la izquierda, es decir, en lugar de adoptar un sistema de respuesta a incidentes, todos son responsables de la seguridad desde el principio, incluso en las etapas de planificación.

El objetivo de este trabajo adopta este enfoque, donde se va a implementar un ciclo de vida DevSecOps abordando la integración continua y las pruebas de seguridad automatizadas como parte del flujo de trabajo.

De esta forma, se va a hacer un seguimiento de problemas de seguridad para garantizar la identificación temprana de cualquier riesgo.

Con ello se pretende demostrar que DevSecOps fusiona la seguridad, el desarrollo y las operaciones para que actúen de forma conjunta y lograr un objetivo común al realizar mejoras en los procesos, herramientas y colaboraciones en equipo.

Palabras clave: DevSecOps, SDLC, Agile, CI, CD, SAST, DAST, SCA, Jenkins

Abstract

The software is complex and vulnerable to numerous security problems. Enterprise culture frequently places security in the final phase of the software development life cycle.

DevSecOps shifts security to the left, i.e. instead of adopting an incident response system, all parties involved are responsible for security from the start, even in the early planning stages.

The objective of this work takes this approach, where a DevSecOps life cycle is going to be implemented where continuous integration and automated security testing are part of the workflow

In this way, security issues will be tracked to ensure early identification of any risks.

This is to demonstrate that DevSecOps merges security, development and operations so that they operate together and achieve a common goal by achieving improvements in the processes, tools and collaboration of work teams.

Keywords: DevSecOps, SDLC, Agile, CI, CD, SAST, DAST, SCA, Jenkins

Índice

1.	Introducción	1
1.1	Implantar un sistema de entrega continua	7
1.2	Control de cambios automatizado	7
1.3	DevOps con Seguridad: integración continua	8
1.3.1	Codificación	9
1.3.2	Compilación	9
1.3.3	Despliegue y Ejecución de pruebas	10
1.3.4	Entrega de software nuevo o actualizaciones	10
2.	Objetivos	10
3.	Estado del arte	10
4.	Fases DevOps	13
4.1	Planificación	14
4.2	Desarrollo o implementación	14
4.3	Integración continua	15
4.4	Despliegue y operación	15
4.5	Monitorización	16
4.6	Feedback o retroalimentación	17
5.	Herramientas	17
6.	Desarrollo del laboratorio	19
6.1	Instalación y configuración de las herramientas	19
6.1.1	GitHub	19
6.1.2	Git	21
6.1.3	SonarQube	22
6.1.4	Maven	24
6.1.5	Jenkins	25
6.1.6	Dependency-Check	30
6.1.7	Docker	31
6.1.8	MailDev	34
6.1.9	OWASP ZAP	38
6.1.10	Jira	40
7.	Resultados obtenidos tras la construcción del proyecto	45
7.1	Jenkins y GitHub	45
7.2	Jenkins y SonarQube	45
7.3	Jenkins y Dependency-Check	48
7.4	Jenkins y Docker	51
7.5	Jenkins y MailDev	54
7.6	Jenkins y OWASP ZAP	55
7.7	Jenkins y Jira	56
8.	Conclusiones	60
8.1	Conclusiones generales	60
8.2	Conclusiones personales	61
9.	Referencias bibliográficas	63

Índice de figuras

<i>Ilustración 1: Desarrollo de Software Agile</i>	2
<i>Ilustración 2: Desarrollo de Software en Cascada</i>	3
<i>Ilustración 3: Ciclo DevOps</i>	4
<i>Ilustración 4: DevOps vs. DevSecOps</i>	5
<i>Ilustración 5: La seguridad de DevOps está automatizada</i>	7
<i>Ilustración 6: Delivery Pipeline Diagram</i>	9
<i>Ilustración 7: Software Development Life Cycle</i>	11
<i>Ilustración 8: 2018 Vulnerabilities by category</i>	12
<i>Ilustración 9: Relative cost to fix bugs, based on time of detection</i>	13
<i>Ilustración 10: DevSecOps</i>	13
<i>Ilustración 11: Security breaches</i>	15
<i>Ilustración 12: Free and Open-Source SIEM Tools</i>	16
<i>Ilustración 13: GitHub</i>	17
<i>Ilustración 14: Git</i>	17
<i>Ilustración 15: Maven</i>	18
<i>Ilustración 16: SonarQube</i>	18
<i>Ilustración 17: Dependency-Check</i>	18
<i>Ilustración 18: Jenkins</i>	18
<i>Ilustración 19: Docker</i>	19
<i>Ilustración 20: MailDev</i>	19
<i>Ilustración 21: ZAP</i>	19
<i>Ilustración 22: Jira</i>	19
<i>Ilustración 23: GitHub – New Repository</i>	21
<i>Ilustración 24: TFM-GitHub/WebGoat</i>	21
<i>Ilustración 25: git remote show origin</i>	22
<i>Ilustración 26: TFM-GitHub/WebGoat Pull request</i>	22
<i>Ilustración 27: Wrapper Java Command</i>	23
<i>Ilustración 28: Start SonarQube</i>	23
<i>Ilustración 29: SonarQube running localhost:9000</i>	24
<i>Ilustración 30: SonarQube Add Project</i>	24
<i>Ilustración 31: Project WebGoat Sonar Jenkins</i>	24
<i>Ilustración 32: Apache Maven</i>	24
<i>Ilustración 33: Jenkins Plugin</i>	25
<i>Ilustración 34: Jenkins running localhost:8080</i>	27
<i>Ilustración 35: Jenkins Plugins Administration</i>	27
<i>Ilustración 36: Jenkins Plugins</i>	28
<i>Ilustración 37: Jenkins - SonarQube Server</i>	28
<i>Ilustración 38: Jenkins – SonarQube Scanner Configuration</i>	28
<i>Ilustración 39: Jenkins – Dependency-Check Install</i>	28
<i>Ilustración 40: Jenkins – Job WebGoat</i>	29
<i>Ilustración 41: Jenkins – Job WebGoat Dashboard</i>	29
<i>Ilustración 42: Jenkins – GitHub Project Url</i>	29
<i>Ilustración 43: Jenkins – GitHub Repository Url</i>	30
<i>Ilustración 44: Jenkins – SonarQube Scanner Analysis Configuration</i>	30
<i>Ilustración 45: Jenkins – Dependency-Check Installation</i>	30
<i>Ilustración 46: Jenkins – Dependency-Check Publish</i>	31
<i>Ilustración 47: Docker Desktop</i>	32
<i>Ilustración 48: Docker Desktop is running</i>	32
<i>Ilustración 49: Docker Desktop Dashboard</i>	33
<i>Ilustración 50: Jenkins – Docker Create Image</i>	33
<i>Ilustración 51: Dockerfile</i>	34
<i>Ilustración 52: Jenkins – Docker Create Container</i>	34
<i>Ilustración 53: Jenkins – Docker Start Container</i>	34

<i>Ilustración 54: Docker Pull MailDev</i>	<i>35</i>
<i>Ilustración 55: Docker – Image & Container MailDev.....</i>	<i>36</i>
<i>Ilustración 56: MailDev – http 1080.....</i>	<i>36</i>
<i>Ilustración 57: Jenkins – Extended E-mail Notification.....</i>	<i>37</i>
<i>Ilustración 58: Jenkins – Extended Test Mail.....</i>	<i>37</i>
<i>Ilustración 59: MailDev – Test Mail.....</i>	<i>37</i>
<i>Ilustración 60: Jenkins – Editable Email Notification.....</i>	<i>38</i>
<i>Ilustración 61: OWASP ZAP proxy port</i>	<i>39</i>
<i>Ilustración 62: Jenkins – OWASP ZAP Configuration.....</i>	<i>40</i>
<i>Ilustración 63: Web Site Atlassian.....</i>	<i>41</i>
<i>Ilustración 64: New Project Atlassian.....</i>	<i>41</i>
<i>Ilustración 65: Board Jira Software</i>	<i>42</i>
<i>Ilustración 66: Default Dashboard Jira</i>	<i>42</i>
<i>Ilustración 67: Recent Activity Jira</i>	<i>43</i>
<i>Ilustración 68: Token Jira.....</i>	<i>43</i>
<i>Ilustración 68: Run Jira Project</i>	<i>44</i>
<i>Ilustración 69: Run Windows Command</i>	<i>44</i>
<i>Ilustración 70: Creating an issue for Jira</i>	<i>44</i>
<i>Ilustración 71: Jenkins – GitHub Output.....</i>	<i>45</i>
<i>Ilustración 72: Jenkins – SonarQube Output</i>	<i>46</i>
<i>Ilustración 73: Jenkins - SonarQube Execution Success.....</i>	<i>46</i>
<i>Ilustración 74: SonarQube Analysis Dashboard.....</i>	<i>47</i>
<i>Ilustración 75: SonarQube Analysis Results</i>	<i>47</i>
<i>Ilustración 76: SonarQube Risk</i>	<i>48</i>
<i>Ilustración 77: SonarQube Security Hotspots</i>	<i>48</i>
<i>Ilustración 78: Jenkins - DependencyCheck Output</i>	<i>49</i>
<i>Ilustración 79: Jenkins - DependencyCheck Trend.....</i>	<i>49</i>
<i>Ilustración 80: Jenkins - DependencyCheck Results</i>	<i>50</i>
<i>Ilustración 81: SonarQube - DependencyCheck Tab More.....</i>	<i>50</i>
<i>Ilustración 82: SonarQube - DependencyCheck Scan Information.....</i>	<i>51</i>
<i>Ilustración 83: SonarQube - DependencyCheck Published Vulnerabilities</i>	<i>51</i>
<i>Ilustración 84: Jenkins – Docker Build Image</i>	<i>52</i>
<i>Ilustración 85: Docker Webgoat Image</i>	<i>52</i>
<i>Ilustración 86: Docker Webgoat Container</i>	<i>53</i>
<i>Ilustración 87: WebGoat Login Page</i>	<i>53</i>
<i>Ilustración 88: WebGoat Navigation.....</i>	<i>54</i>
<i>Ilustración 89: Jenkins – Output MailDev.....</i>	<i>54</i>
<i>Ilustración 90: MailDev Execution Successful.....</i>	<i>55</i>
<i>Ilustración 91: Jenkins - Output Zap</i>	<i>55</i>
<i>Ilustración 92: ZAP Scanning Report</i>	<i>56</i>
<i>Ilustración 93: Run Jira</i>	<i>57</i>
<i>Ilustración 94: Jenkins Output Jira</i>	<i>57</i>
<i>Ilustración 95: Jira Notification</i>	<i>58</i>
<i>Ilustración 96: Jira App</i>	<i>58</i>
<i>Ilustración 97: Jira Detail App</i>	<i>59</i>

Introducción y objetivos

1. Introducción

El ciclo de vida del desarrollo de *software*, conocido por las siglas en inglés SDLC o *Systems Development Life Cycle*, es un proceso de construcción o mantenimiento de sistemas de *software* y representa las diferentes fases que, por lo general, incluyen desde el análisis preliminar hasta las pruebas y evaluación posteriores al desarrollo del *software*.

Este proceso incorpora los modelos y metodologías que los equipos de desarrollo utilizan para desarrollar *software*, metodologías que constituyen el marco para planificar y controlar todo el proceso de desarrollo.

Una aplicación o un sistema de información está diseñado para realizar un conjunto específico de tareas que proporcionan resultados que implican cálculos y procesamiento complejos. En consecuencia, resulta un trabajo arduo y tedioso gestionar todo el proceso de desarrollo para garantizar que el producto final presente un alto grado de integridad y robustez, y responda satisfactoriamente a las pruebas de aceptación de usuario que verifican que el *software* desarrollado está listo para ser lanzado al mercado.

Actualmente, hay dos metodologías SDLC que son utilizadas por la mayoría de los desarrolladores de *software*, la metodología tradicional y metodología ágil.

En el ciclo de vida de desarrollo tradicional, los desarrolladores y sus equipos suelen fijar reuniones con otros equipos involucrados en el proceso SDLC con el objetivo de detallar los requisitos funcionales y de diseño previos al comienzo de la implementación.

A la fase de diseño le sigue la fase de codificación. La fase de pruebas tiene lugar cuando se completa todo el proceso de codificación y solo se presenta el producto final a las partes interesadas después de que en estas pruebas no se detecte ningún problema.

Una de las desventajas de esta metodología tradicional es que los equipos construyen el sistema de manera "única". En el supuesto de que surja un problema durante la fase de pruebas, lo peor de este escenario es que todo el módulo/desarrollo tiene que revertirse para rectificar ese problema.

Otro inconveniente del SDLC tradicional es que, en la mayoría de los casos, las

partes interesadas no conocen a priori lo que realmente quieren implementar en el sistema, por lo tanto, el modelo de requisitos diseñado en las fases anteriores puede no cumplir con las características reales que deben implementarse.

Las solicitudes de cambio de los usuarios o partes implicadas pueden establecerse después de que el producto final sea presentado y lanzado al mercado y este cambio puede causar varios problemas de compatibilidad e integridad del *software*.

Con todos estos inconvenientes, surge la necesidad de establecer un proceso iterativo donde los cambios puedan realizarse de forma más ágil. Es en este punto dónde se crea la metodología ágil para los procesos de desarrollo de *software* donde el cliente está presente en todas las fases de desarrollo. Esta metodología facilita la interacción entre todas las partes implicadas ya que se pone el foco en las personas y no en los procesos, permitiendo dimensionar los proyectos de forma más eficiente minimizando los riesgos (What is Agile, 2020).

En todo caso, cada metodología de desarrollo de *software* es más funcional con ciertos tipos de proyectos. Los diferentes tipos de complejidades en un proyecto exigen distintos análisis y niveles de experiencia en el método de elección.

Teniendo en cuenta que ningún método es cien por cien perfecto, los desarrolladores tienen que analizar todos los pros y los contras. Un conjunto de factores a considerar son el presupuesto, el alcance del proyecto, los recursos disponibles, el marco de tiempo y las preferencias.

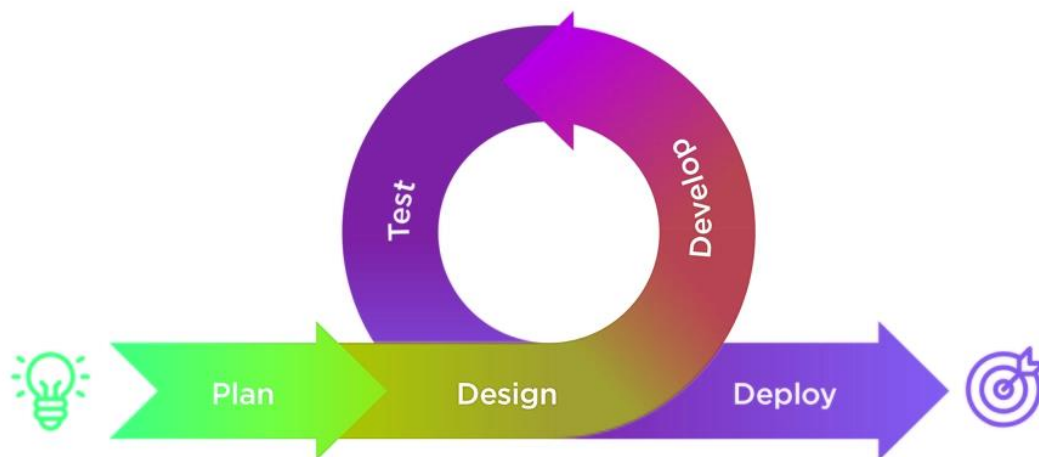


Ilustración 1: Desarrollo de Software Agile



Ilustración 2: Desarrollo de Software en Cascada

A continuación, se muestra una tabla comparativa de las características más importantes de la metodología tradicional, como es el modelo de desarrollo en cascada, y la metodología ágil.

Agile vs. Cascada

	Agile	Cascada
Poner el foco en	Entrega del producto lo más rápido posible	Mantenerse dentro del presupuesto y los plazos establecidos
Proceso de desarrollo	Todas las fases forman parte del ciclo (desarrollo iterativo)	Lineal: una fase tras otra
Ritmo de desarrollo	Iteraciones de 2 a 4 semanas	Desarrollo sin <i>feedback</i> del cliente
Involucramiento del cliente	A lo largo del proyecto	En hitos
Alcance	Adaptación a los nuevos cambios del mercado que mejoran el producto	Detallado desde el principio
Equipo	Se prefieren equipos más pequeños con estrecha comunicación entre sus miembros (equipos autoorganizados)	Organización jerárquica con roles claramente definidos; no existe mucha comunicación

Sin embargo, la metodología Agile no resuelve el problema de comunicación entre los diferentes elementos que conforman el proceso de desarrollo de un sistema *software*: el equipo de desarrollo y el equipo de operaciones.

Es por ello, que aparece el modelo DevOps como respuesta al modelo tradicional de desarrollo de *software* que exigía que quienes escriben código

se separen organizativa y funcionalmente de aquellos que despliegan y soportan ese código. Este conjunto de prácticas permite automatizar e integrar los procesos entre el desarrollo de *software* y los equipos de TI, para que puedan construir, probar y lanzar *software* de manera más rápida y confiable. Los desarrolladores y los profesionales de TI/Operaciones tradicionalmente tenían objetivos diferentes (y a menudo competitivos), liderazgo definido por departamento, indicadores clave de rendimiento (KPIs) diferenciados, y a menudo trabajaban en distintas ubicaciones o incluso en edificios separados. El resultado es el de equipos aislados, largas horas de trabajo, lanzamientos fallidos y clientes insatisfechos.

El término DevOps se conforma combinando las palabras "desarrollo" y "operaciones" y supone un cambio cultural que cierra la brecha entre los equipos de desarrollo y operación (DevOps, s.f.).

DevOps no es simplemente un proceso o un enfoque diferente para el desarrollo, es un cambio de cultura que implica un cambio de mentalidad, una mejor colaboración y una integración más estrecha.

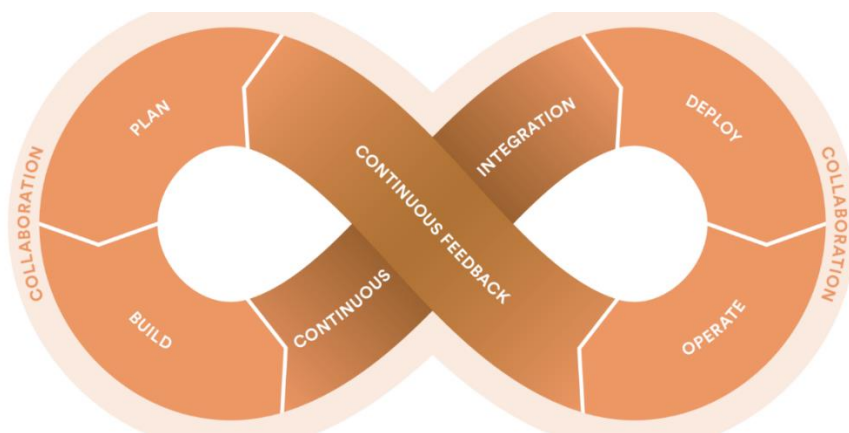


Ilustración 3: Ciclo DevOps

Si bien el modelo DevOps se ha convertido en la corriente principal, la mayoría de las empresas no han adaptado sus modelos de seguridad y sus equipos de seguridad se quedan rezagados tratando de mantener el ritmo de los ciclos ágiles de desarrollo.

Es en este punto dónde cobra especial relevancia el modelo DevSecOps que

integra la seguridad en el proceso DevOps ayudando a prevenir y abordar los riesgos de seguridad a medida que aparecen en el ciclo de desarrollo (DevSecOps, s.f.).

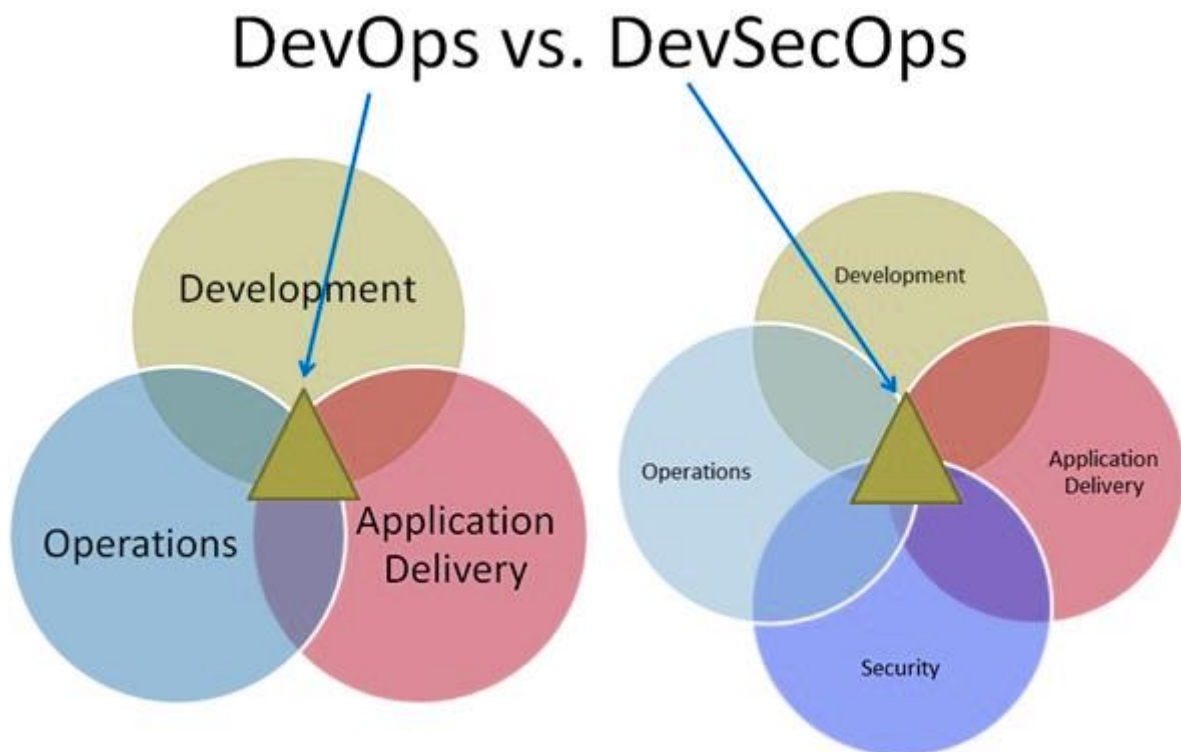


Ilustración 4: DevOps vs. DevSecOps

Para lograr un proceso DevOps exitoso y seguro, las prácticas de desarrollo seguro deben integrarse en cada fase del ciclo de vida de DevOps, desde el inicio hasta el mantenimiento del *software*.

Este tipo de seguridad incorporada a DevOps tiene como objetivo incluir una cultura y prácticas de seguridad en todo el flujo de trabajo de DevOps a través de una mejor colaboración y responsabilidad compartida, dando como resultado un lanzamiento de producto más rápido y seguro.

En esencia, DevSecOps ha cambiado la naturaleza misma de cómo se debe implementar la seguridad de las aplicaciones y hace referencia a la seguridad integrada y no su perímetro de seguridad.

Bajo DevOps, algunas organizaciones de desarrollo hacen lanzamientos de *software* quincenales, semanales e incluso diarios, pero se enfrentan a modelos de seguridad de aplicaciones más antiguos. Como resultado, las pruebas funcionales de desarrollo están descompensadas respecto a las

pruebas de seguridad: no es efectivo realizar auditorías de código estático (SAST) en *software* productivo o no es posible realizar análisis de código dinámico (DAST) sin automatizarlo en *software* que se publique semanalmente. Por lo que las organizaciones necesitan una garantía de seguridad en su SDLC que sea efectiva, rápida, y, sobre todo, automatizada.

La automatización de estas tareas programadas desempeña un papel clave para permitir que la seguridad de las aplicaciones mantenga los ritmos de lanzamiento de *software* que se producen cada día, semana o quincena, así como para aumentar la agilidad general dentro del SDLC.

Esta agilidad hace posible que los desarrolladores de código conozcan los resultados de los análisis estáticos de código, conocidos por las siglas en inglés SAST, *Static Application Security Testing*. Dichas pruebas tienen como objetivo medir la seguridad de las prácticas de desarrollo para encontrar vulnerabilidades en el código fuente que puedan hacer que una aplicación sea susceptible a ataques (Blazquez, 2020).

En lugar de dar a los desarrolladores resultados SAST trimestrales, la automatización proporciona análisis de código a los desarrolladores en tiempo real, ya que están chequeando el código de forma continua (a través de ciertos *plugins* integrados en los IDE, *Integrated Development Environment* o entornos de desarrollo integrado).

Presentar los resultados del análisis de código a los desarrolladores, mientras siguen desarrollando código, tiene varias ventajas:

- los problemas de codificación se detectan antes y con mayor facilidad dado que los desarrolladores todavía tienen muy presente la funcionalidad del código.
- mitiga el riesgo de que los desarrolladores no recuerden el contexto del código en una fecha posterior.
- ayuda a reducir tiempo de resolución de fallos de seguridad y vulnerabilidades de código fuente permitiendo ahorrar costes de recursos económicos y humanos.



Ilustración 5: La seguridad de DevOps está automatizada

Para mejorar los procesos de gestión de cambios en el código y agilizar el lanzamiento de nuevas entregas de producto, garantizando la estabilidad del entorno de trabajo, se hace necesaria la automatización de tareas repetitivas que van a permitir lanzar versiones de *software* con mayor frecuencia y de mejor calidad.

1.1 Implantar un sistema de entrega continua

El riesgo en el rendimiento del desarrollo de *software* es mayor para cambios de código de mayor volumen. Generalmente, está estrechamente relacionado con las dependencias entre diferentes módulos de *software* que orquestan todo un aplicativo. Esto lleva implícito mucha planificación, importantes gastos generales de proceso y equipos de TI que pasan largas horas de trabajo durante el lanzamiento a producción. También precisa un plan detallado de reversión que es igualmente complicado.

De la misma forma, los retrasos en la entrega de estos cambios conllevan mayores riesgos y costes más altos. Aquí es exactamente donde las prácticas de DevOps son un factor determinante ya que permiten que los equipos entreguen conjuntos de cambios a producción con mayor frecuencia, con menores riesgos y costes generales. Si la entrega de cambios es más frecuente, tendrá una mayor repetibilidad y hará que todos los involucrados tengan más confianza en el proceso.

1.2 Control de cambios automatizado

El control de cambios tradicional requiere aprobaciones y verificaciones manuales. Muchas de estas verificaciones manuales se pueden lograr de forma automática y segura mediante la incorporación de lógica de negocio, umbrales

predefinidos y controles automatizados en todo el proceso de entrega.

Como ejemplos, las aprobaciones del diseño de *software* requieren de pruebas de análisis de código estático y verificación de conjuntos de cambios; pruebas de regresión, pruebas de rendimiento; etc. Casi todas pueden automatizarse en muchos niveles diferentes dependiendo de la madurez de DevOps de una organización.

La incorporación de estas aprobaciones de manera automatizada permite una implementación continua que supone un beneficio fundamental del modelo DevOps.

Además, el riesgo está bien administrado en este modelo, ya que la automatización de la gestión de cambios incorpora todas las comprobaciones de seguridad y auditoría que proporciona un proceso formal de aprobación de cambios.

1.3 DevOps con Seguridad: integración continua

La mayoría de las organizaciones realizan una inversión importante de gasto en TI en las áreas de infraestructura y seguridad de red. Sin embargo, el gasto en la seguridad del *software* es excesivamente bajo.

Si bien asegurar la infraestructura y la red es absolutamente necesario, el enfoque en la seguridad del *software* debe abordarse igualmente en las organizaciones TI actuales.

La implementación de un *delivery pipeline* permitirá a las organizaciones garantizar un mejor control de seguridad y cumplimiento continuo y menores riesgos en el proceso de desarrollo.

La siguiente tabla muestra las distintas fases de automatización y control de seguridad del ciclo de vida completo de una aplicación a través de un *pipeline*: codificación, compilación, ejecución de *tests* y despliegue e implementación del proceso de lanzamiento.

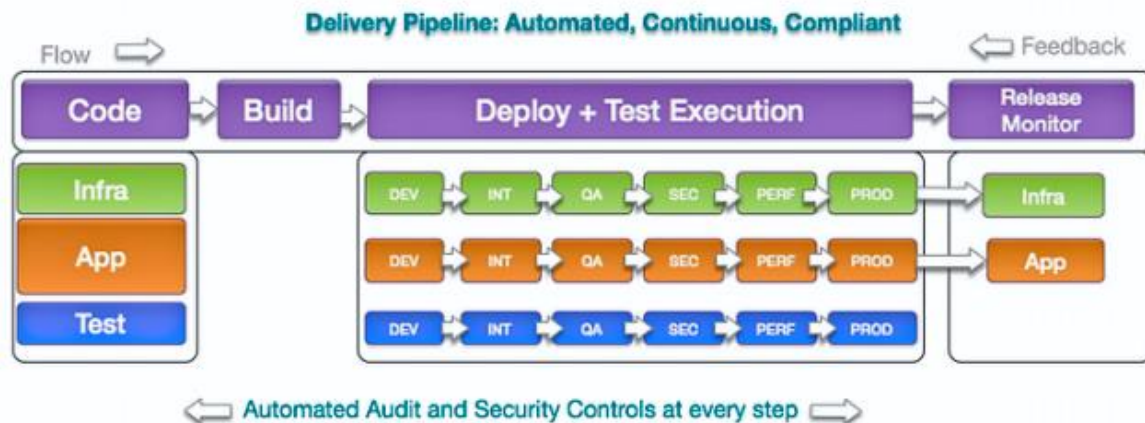


Ilustración 6: Delivery Pipeline Diagram

1.3.1 Codificación

Incorporar seguridad en el código es aplicar buenas prácticas de desarrollo seguro como parte esencial de los flujos de trabajo en la cultura DevOps. En integración continua, cada vez que un desarrollador registra un cambio de código, el sistema se construye/compila y prueba de forma automática, lo que proporciona *feedback* de forma rápida y frecuente sobre el estado del código desarrollado.

Escribir código seguro supone realizar revisiones de código y utilizar herramientas automatizadas de análisis estático para detectar errores de codificación comunes y debilidades que puedan llegar a suponer una vulnerabilidad en el aplicativo. Estas herramientas también pueden encontrar errores lógicos que podrían provocar fallos en tiempo de ejecución.

El código de la aplicación debe analizarse en busca de debilidades de seguridad y otros defectos de *software* durante un ciclo de integración continua.

1.3.2 Compilación

El código de la aplicación es compilado, es decir, los binarios son escaneados y analizados. Las herramientas SAST que soportan este tipo de análisis, buscan encontrar *dataflows* en la lógica de la aplicación y reportar esos posibles *findings* encontrados en dichos flujos.

Las librerías de terceros (incluidas las librerías de código abierto u *open source*) deben analizarse en busca de vulnerabilidades de seguridad y tipo de licencia de uso. Es muy importante que estas librerías, al ser usadas como dependencias en los desarrollos, se mantengan actualizadas y parcheadas; por

lo que es una buena práctica incluir su revisión en lo que se conoce como SCA o *Software Composition Analysis*.

1.3.3 Despliegue y Ejecución de pruebas

El proceso de despliegue debe ser automatizado y/o programado para su reproducción. Cada *deployment* debe ir seguido de una ejecución de pruebas automatizadas para verificar que los cambios no afectan al funcionamiento de la aplicación. Finalmente, los resultados de las pruebas deben almacenarse para garantizar la trazabilidad y ayudar a solucionar problemas.

1.3.4 Entrega de software nuevo o actualizaciones

El proceso de lanzamiento de *software* (*release*) en una empresa grande, generalmente implica mucha planificación, procesos de aprobación y despliegue manual de nuevo *software* en entornos de producción. Este proceso de *release* tradicional comúnmente conlleva un alto riesgo que puede afectar al alcance de la versión a entregar.

El modelo DevOps, con un ciclo de despliegue más corto y rápido, facilita el proceso de liberación de nuevas *releases* reduciendo estos riesgos.

2. Objetivos

El objetivo principal de este trabajo de fin de máster es la definición y puesta en práctica de un ciclo de vida de desarrollo seguro de un sistema o *software*.

Para llevar a cabo esta tarea, se va a implementar un sistema de despliegue continuo, y se va a incluir la seguridad de forma automática en todas las fases del ciclo integrando todo en un mismo *job*.

- Diseño del ciclo de vida del desarrollo seguro de *software*.
- Implementación de un sistema de automatización y despliegue continuo.
- Incorporación de pruebas de seguridad en todas las fases del ciclo de vida del *software*.

3. Estado del arte

El SDLC asienta las bases de las diferentes etapas por las que pasa el desarrollo de *software* en una organización, desde que se inicia su planificación a partir de un requerimiento hasta que finaliza una vez está implantado en producción.

Estas cinco etapas [figura 7] son: la planificación o definición de requisitos del *software*, análisis, diseño técnico, implementación o codificación y fase de pruebas e integración y despliegue y mantenimiento.

Se trata de un proceso de desarrollo cíclico que tiene como objetivo fundamental construir un *software* funcional y de calidad que se ajuste a la demanda del cliente según sus necesidades, mediante la entrega en los plazos programados.

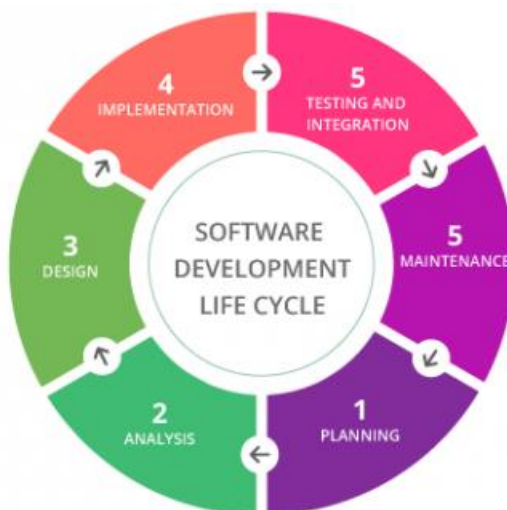


Ilustración 7: Software Development Life Cycle

Atendiendo a las distintas fases, las metodologías de desarrollo se han ido adaptando a las exigencias del mercado logrando que los procesos de negocio sean mucho más flexibles con entregas de producto en menor tiempo y ajustándose a las estimaciones de costes del proyecto.

De igual forma, en los últimos años, el número de vulnerabilidades descubiertas ha crecido considerablemente como consecuencia del acceso sin precedentes a la información y a los activos informáticos.

Algunos factores que aumentan la superficie de ataque son una mayor virtualización (a través de contenedores de aplicaciones), la aparición de dispositivos inteligentes e IoT, y la computación en la nube, entre otros (Research Report, 2019).

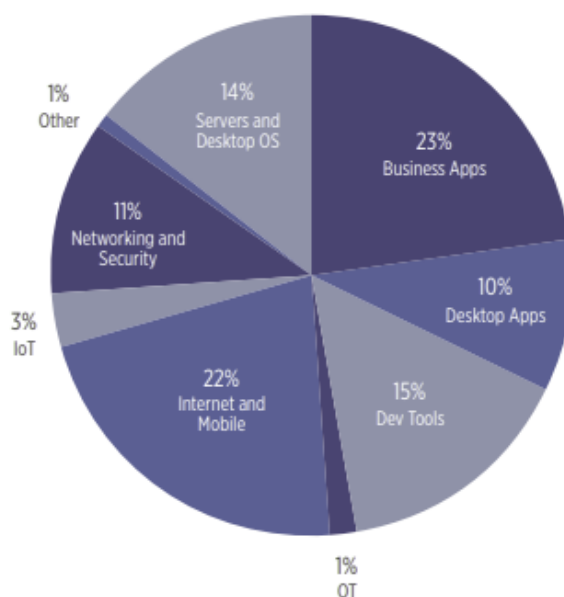


Ilustración 8: 2018 Vulnerabilities by category

No obstante, es frecuente que los departamentos de desarrollo de muchas organizaciones no apliquen prácticas de seguridad por desidia o falta de concienciación, con pretextos del tipo "nuestra aplicación no es un blanco para los ciberdelincuentes", "utiliza protocolo *https*", "está protegida por un *firewall*", y un largo etcétera.

Existe la necesidad de incorporar la seguridad y eliminar la creencia de que la aplicación de estas medidas de seguridad debe introducirse en la última etapa del desarrollo de un producto y cambiar la cultura de concienciación asumiendo que la seguridad es imprescindible desde el inicio del SDLC.

Y esto es así porque el coste de solucionar cualquier problema de seguridad es mayor cuanto más tarde se detecte, como se muestra en el siguiente gráfico (On Point Technology, LLC, n.d.).

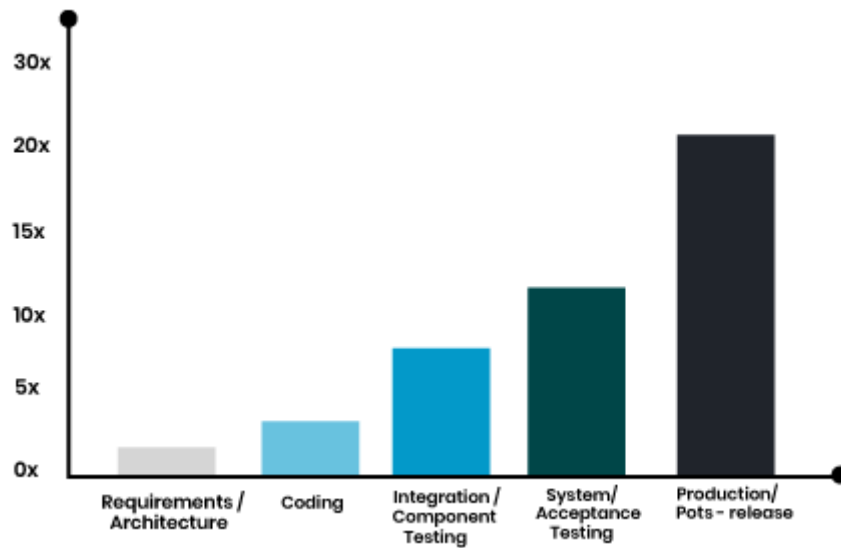


Ilustración 9: Relative cost to fix bugs, based on time of detection

Por tanto, la adopción de prácticas de seguridad en la filosofía DevOps es un elemento imperativo para garantizar la construcción segura (confidencialidad, disponibilidad e integridad) del *software*.

DevOps + Security: DevSecOps

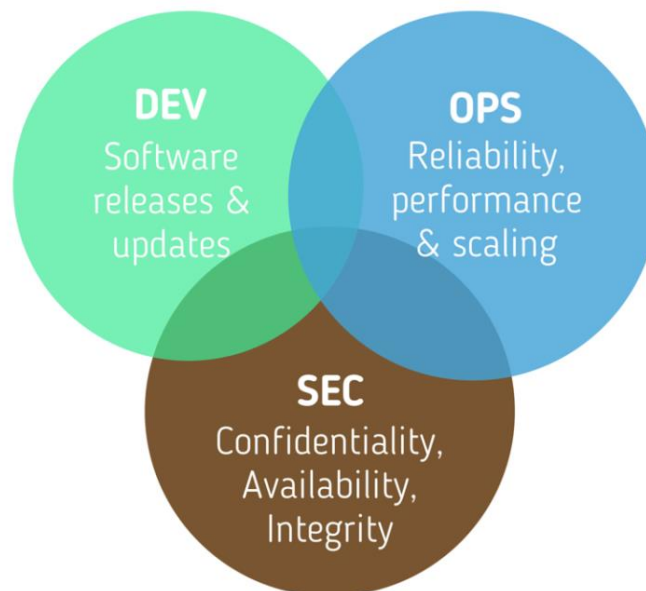


Ilustración 10: DevSecOps

4. Fases DevOps

Para implementar una solución que incluya la seguridad en el SDLC, en primer lugar, se tienen que definir cada una de las fases por las que pasa el desarrollo

de *software* en una organización (EC-Council, 2020).

4.1 Planificación

En esta primera etapa, se diseña la interfaz y la funcionalidad de la aplicación (se incluyen tareas de diseño y arquitectura y tareas de programación más específicas), se establecen criterios de aceptación (definen si las historias de usuario han sido desarrolladas atendiendo a las expectativas del cliente), se evalúan los riesgos y se construyen modelos de defensa contra amenazas.

Un modelo de amenazas eficiente analiza los posibles escenarios de ataque, traza el flujo de datos sensible dentro de la aplicación (autenticación, autorización, algoritmos de cifrado), encuentra mitigaciones para las amenazas y se repite cada vez que hay un cambio en el diseño de la aplicación o una nueva característica.

Además del modelado de amenazas, la guía del proyecto *OWASP Web Security Testing Guide* (WSTG) es una valiosa fuente de información para identificar los requisitos de seguridad y ayudar a realizar un desarrollo seguro de *software*.

Para llevar a cabo todas estas recomendaciones y definir los requerimientos de control de seguridad, se hace necesario educar o concienciar al equipo involucrado en materia de seguridad para desarrollar un marco de trabajo común en toda la organización.

4.2 Desarrollo o implementación

En esta fase, los desarrolladores crean código dentro de un sistema de control de versiones. Este sistema lleva a cabo un registro de todos los cambios realizados en un fichero o conjunto de ficheros a lo largo del tiempo permitiendo recuperar versiones concretas a posteriori.

Las pruebas de seguridad de aplicaciones estáticas (*Static Application Security Testing* o SAST) permiten identificar debilidades de seguridad del código fuente y eliminarlas antes de que se complete el ciclo de vida de desarrollo de *software*. Adicionalmente, se aplica la tecnología conocida como análisis de composición de *software* (*Software Composition Analysis* o SCA) para examinar e identificar vulnerabilidades publicadas en componentes de terceros y librerías que usa la aplicación.

La integración de herramientas SAST al proceso de desarrollo de *software* es un paso crucial ya que hace posible que el analista de seguridad pueda escanear el código fuente y auditar los resultados obtenidos.

Según el Departamento de Seguridad Nacional de los EE.UU., el 90% de las brechas de seguridad se producen debido a vulnerabilidades en el código.

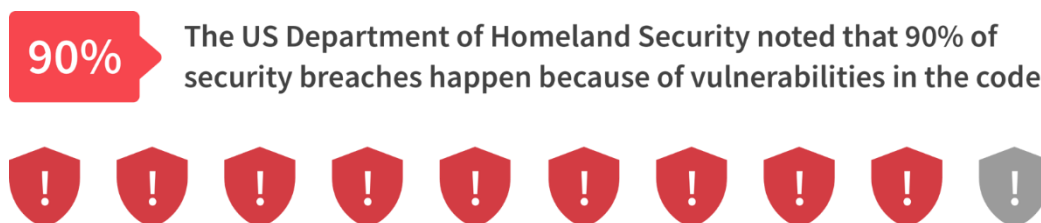


Ilustración 11: Security breaches

Es en esta etapa del proyecto donde se va a llevar a cabo la subida del código fuente de una aplicación pública a un repositorio centralizado.

4.3 Integración continua

Es la fase donde tienen lugar las pruebas automáticas de código.

Estas pruebas se realizan en el repositorio central que aloja el código fuente desarrollado y consisten en la compilación y ejecución de versiones de código y la publicación del *software* para detectar errores y corregirlos de forma temprana, mejorar la calidad del código y reducir los tiempos de validación de las nuevas versiones o actualizaciones que se vayan subiendo al repositorio. Las tareas se realizan de forma periódica, una o varias veces al día.

En referencia al modelo planteado, el código fuente localizado en el repositorio se descargará para realizar el análisis SAST y las dependencias de terceros integradas en el proyecto.

Se evaluará la calidad y seguridad del código analizado en función de unos umbrales definidos en la herramienta SAST y si no se ajusta a las métricas establecidas no se pasará a la siguiente fase de empaquetado y despliegue.

4.4 Despliegue y operación

En esta fase, se lleva a cabo la construcción del *software* ubicado en el repositorio para obtener una versión funcional del mismo. Esta nueva versión del aplicativo será desplegada en el entorno correspondiente (desarrollo,

preproducción, producción).

Se va a crear una imagen y desplegar en un contenedor Docker a partir del código con el objetivo de poder ejecutar la aplicación *software* en cualquier entorno o máquina destinada a tal efecto.

A continuación, se realizan las pruebas de *testing* y calidad de código para hacer un seguimiento, identificar posibles incidentes y resolver los problemas que vayan apareciendo. De esta forma, se asegura que la nueva *release* cumple con los requerimientos definidos en las especificaciones del *software*.

4.5 Monitorización

El proceso de *logging*, monitorización y alertas cubre la gestión del estado de seguridad de una aplicación. Esto incluye capturar qué eventos han ocurrido (*logging*), proporcionar información sobre esos eventos (monitorización) e informar a las partes implicadas cuando existan problemas por resolver (alerta). Las herramientas SIEM (*Security Information and Event Management*) proporcionan una visión global de la seguridad de los activos de información de una organización (Open Source SIEM tools, n.d.).





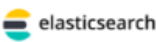




OSSIM		Offers both server-agent and serverless modes, with log analysis for mail servers, databases, and more.
Sagan		Real-time log analysis and correlation tool that's compatible with graphic consoles like Snorby and EveBox.
Splunk Free		Free version of Splunk tool that lets you index up to 500 MB daily for real-time data indexing and alerts.
Snort		Analyzes network traffic in real time, but features make it best-suited for experienced IT professionals.
Elasticsearch		Combine log search types and easily scan through large volumes of logs with this basic tool.
MozDef		A microservices-based tool that can integrate with third-party platforms for straightforward security insights.
ELK Stack		Combines Elasticsearch with tools like Kibana, Beats, and Logstash, for a fuller SIEM solution.
Wazuh		An on-premises tool that offers threat detection, incident response, and compliance support.
Apache Metron		Combines security operations center functions into one centralized, dynamic tool for catching threats.

Ilustración 12: Free and Open-Source SIEM Tools

4.6 Feedback o retroalimentación

Los ciclos de retroalimentación son la esencia de cualquier mejora del proceso Devops y esto es aplicable a prácticas de CI/CD (*Continuous Integration and Continuous Delivery*), monitorización de la infraestructura y desarrollo de aplicaciones.

El ciclo de vida de la aplicación se integrará en un *job* de Jenkins para realizar un seguimiento de todas y cada una de sus fases.

5. Herramientas

Para llevar a cabo este proyecto se van a utilizar las siguientes herramientas de código abierto (*Open Source*):

- **GitHub**: el código fuente de la aplicación se va a alojar en este repositorio posibilitando llevar el control de cambios sobre este código.



Ilustración 13: GitHub

- **Git**: es el sistema de control de versiones más utilizado en el mundo y se va a usar para subir el código fuente al repositorio GitHub a través de la línea de comandos.



Ilustración 14: Git

- **Maven**: herramienta utilizada para la compilación y gestión de proyectos basados en tecnología Java.



Ilustración 15: Maven

- **SonarQube:** plataforma para evaluar la calidad del código fuente de la aplicación.



Ilustración 16: SonarQube

- **OWASP Dependency-Check:** herramienta para detectar vulnerabilidades en componentes de terceros o dependencias del proyecto.



Ilustración 17: Dependency-Check

- **Jenkins:** la integración continua se va a realizar a través de esta herramienta, lo que va a permitir conocer el estado del *software* en cada momento.



Jenkins

Ilustración 18: Jenkins

- **Docker:** el uso del contenedor Docker permite compartir la aplicación y sus dependencias entre distintos entornos y distintos equipos (desarrollo, operaciones).



Ilustración 19: Docker

- **MailDev:** herramienta para enviar correos electrónicos durante la fase de desarrollo y tiene una interfaz web fácil de usar.



Ilustración 20: MailDev

- **OWASP Zed Attack Proxy (ZAP):** herramienta para realizar *pentest* (pruebas de penetración) en aplicaciones web.



Ilustración 21: ZAP

- **Jira:** *software* para la gestión de proyectos de desarrollo.



Ilustración 22: Jira

6. Desarrollo del laboratorio

6.1 Instalación y configuración de las herramientas

6.1.1 GitHub

GitHub es un repositorio de alojamiento de código fuente que proporciona a los

desarrolladores herramientas para la gestión de proyectos de *software* como la descripción de requisitos de *software*, seguimiento de errores, tableros scrum/kanban, *pull requests*, *wikis* de revisión de código, etcétera.

Cuenta con una comunidad de 15 millones de desarrolladores y un ecosistema con cientos de integraciones.

El flujo de trabajo de GitHub es un *workflow* ligero basado en ramas (*branches*) creado en torno a los comandos principales de Git utilizados por equipos de todo el mundo.

Git para Windows proporciona una emulación BASH (shell predeterminada en Linux y macOS) que se utiliza para ejecutar Git desde la línea de comandos.

Para la realización de este laboratorio se ha instalado la versión para Windows de la herramienta Git Bash desde la página oficial (git).

La aplicación web que se ha elegido para este proyecto es la aplicación WebGoat de OWASP. Se trata de una aplicación vulnerable por defecto e implementada de forma deliberada para permitir a los desarrolladores interesados probar las vulnerabilidades que se encuentran en aplicaciones basadas en Java y que utilizan componentes *open source* comunes y populares (OWASP WebGoat, s.f.).

El objetivo principal del proyecto WebGoat es simple: crear un entorno de aprendizaje destinado a realizar pruebas de seguridad en las aplicaciones web. Se trata de un proyecto o *software* libre alojado en el repositorio GitHub (GitHub, n.d.).

Se descarga el proyecto al entorno local con la intención de crear el repositorio en GitHub y poder compilarlo a través de una tarea o *job* de Jenkins.

Se realiza el registro en GitHub y se crea un repositorio haciendo '*clic*' en el icono + de la parte superior derecha.

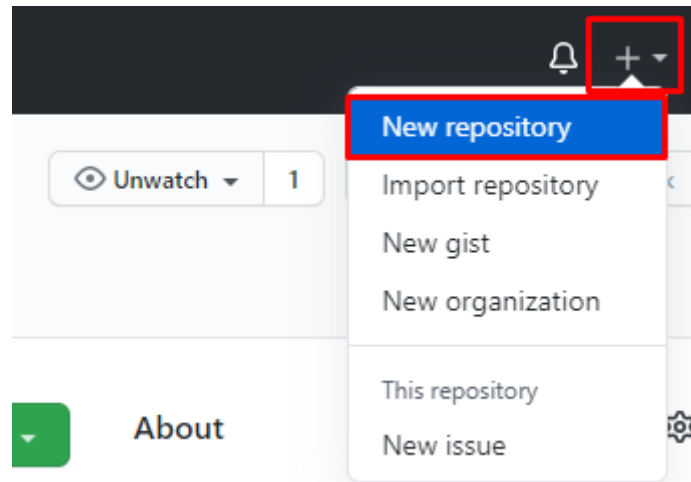


Ilustración 23: GitHub – New Repository

Se crea el proyecto WebGoat para alojar el código que se va a compilar y analizar en la cuenta TFM-GitHub (GitHub, s.f.).

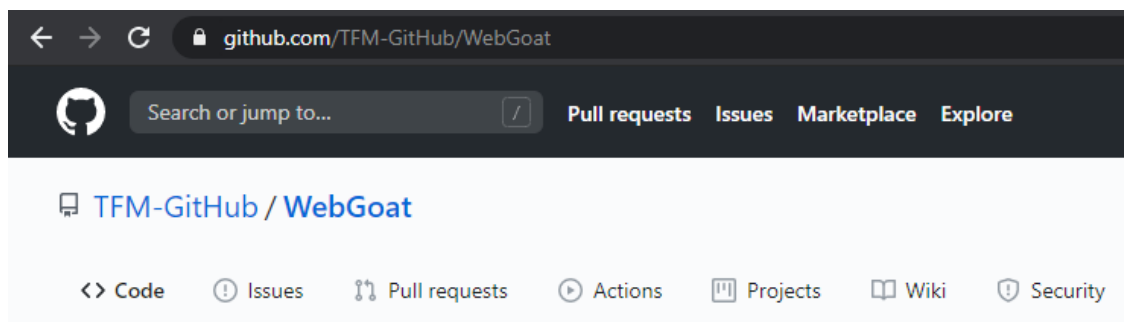


Ilustración 24: TFM-GitHub/WebGoat

6.1.2 Git

Una vez instalado Git se abre una terminal desde la carpeta del proyecto y se ejecuta la instrucción:

git init

Este comando creará un nuevo subdirectorio . ***git*** en el directorio de trabajo actual, si todavía no lo tiene.

A continuación, se ejecuta el comando ***git add .*** para añadir de forma recursiva todos los archivos que se encuentren en el directorio principal.

Y se lanza el comando “***commit***” para enviar todos los cambios al repositorio local:

git commit -m 'commit v1.0'

El siguiente comando permite crear una conexión remota con un repositorio nuevo:

git remote add origin https://github.com/TFM-GitHub/WebGoat.git

Por último, se hace el “push” para cargar contenido del repositorio local al repositorio remoto:

git push -u origin master

El comando ***git remote show origin*** muestra la *url* del repositorio remoto y la información del rastreo de ramas:

```
José María Acuña@LAPTOP-VLTR6BKP MINGW64 ~/.jenkins/workspace/WebGoat ((69b0a2d...))
$ git remote show origin
* remote origin
  Fetch URL: https://github.com/TFM-GitHub/WebGoat.git
  Push URL: https://github.com/TFM-GitHub/WebGoat.git
  HEAD branch: master
  Remote branch:
    master tracked
```

Ilustración 25: git remote show origin

Si el proceso se ha ejecutado de forma satisfactoria, se accede al repositorio en GitHub y se observa que el proyecto se ha subido de forma correcta.

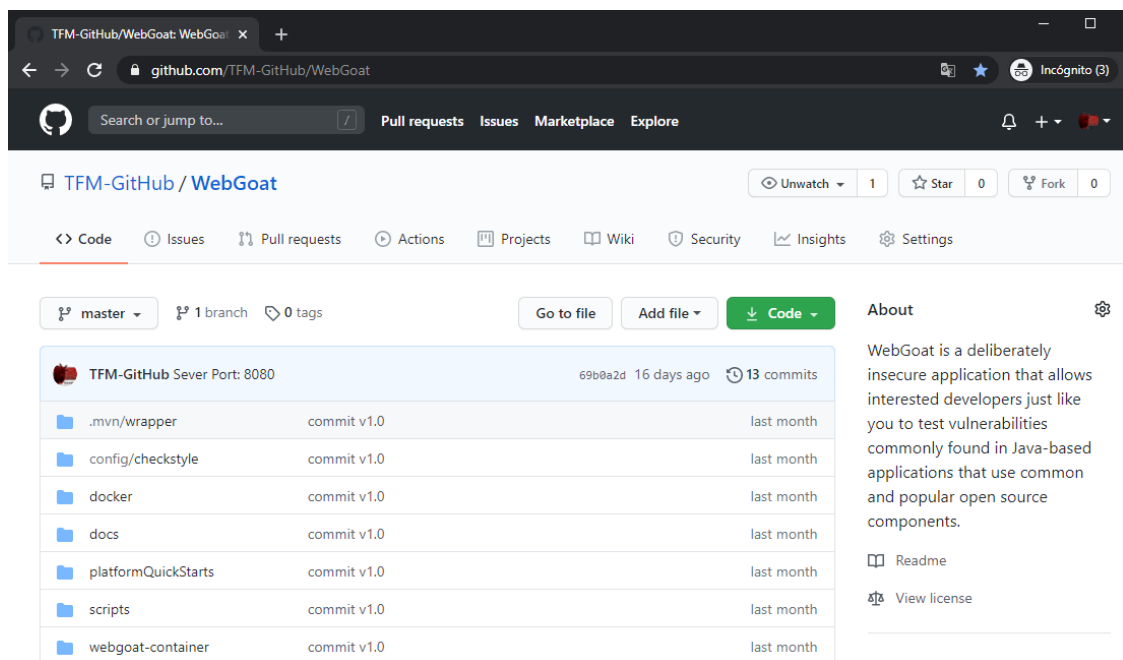


Ilustración 26: TFM-GitHub/WebGoat Pull request

6.1.3 SonarQube

SonarQube permite realizar una evaluación estática del código fuente del aplicativo desde el punto de vista de la seguridad y calidad sin necesidad de ejecutarlo.

Para la instalación de SonarQube, se ha descargado la aplicación desde el repositorio oficial (SonarQube, n.d.) además de tener que descargar también el escáner o analizador de ficheros fuentes para analizar e identificar la tecnología usada en el aplicativo (SonarScanner, n.d.).

SonarScanner se puede ejecutar de forma muy sencilla por línea de comandos a partir de una serie de instrucciones relativas a la tecnología con la que se ha implementado el proyecto (Gradle, MSBuild, Maven, Jenkins, Azure DevOps o Ant) ya que cuenta con un script para Windows, Mac y Linux.

Para este proyecto fin de máster, se va a analizar el código alojado en el repositorio GitHub haciendo uso de la herramienta Jenkins para descargarlo y automatizar la inspección.

SonarQube levanta un servidor web en el puerto 9000. No obstante, desde el fichero de propiedades “*sonar.properties*” se pueden modificar determinados parámetros en función de las necesidades de análisis del aplicativo, se pueden añadir permisos para la creación de tablas, modificar el puerto por defecto dónde escucha la base de datos embebida H2, cambiar el esquema *jdbc* (*Java DataBase Connection*) de Oracle, PostgreSQL o SQLServer si se usa una de estas bases de datos, indicar otro puerto HTTP de conexión y un largo etcétera.

En el fichero “*wrapper.conf*” ubicado en el directorio *conf* se tiene que especificar la ruta dónde se encuentra el JDK en el equipo.

```
1 # Path to JVM executable. By default it must be available in PATH.
2 # Can be an absolute path, for example:
3 #wrapper.java.command=/path/to/my/jdk/bin/java
4 wrapper.java.command=C:\Program Files\Java\jdk-11.0.8\bin\java
```

Ilustración 27: Wrapper Java Command

Se ejecuta el archivo *StartSonar.bat* del directorio */bin* y se accede a la instancia local en el puerto 9000.

```
C:\TFM-Deloitte\sonarqube-8.4.2.36762\bin\windows-x86-64>StartSonar.bat
wrapper | --> Wrapper Started as Console
wrapper | Launching a JVM...
jvm 1   | Wrapper (Version 3.2.3) http://wrapper.tanukisoftware.org
jvm 1   | Copyright 1999-2006 Tanuki Software, Inc. All Rights Reserved.
```

Ilustración 28: Start SonarQube

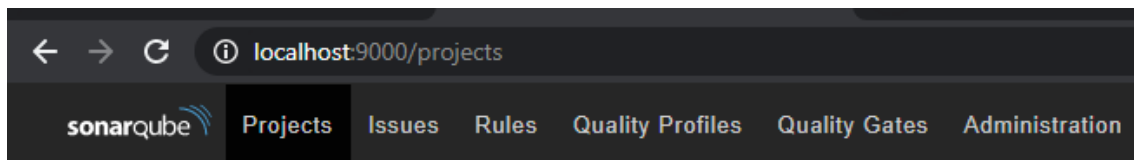


Ilustración 29: SonarQube running localhost:9000

A continuación, se crea un proyecto desde el icono + de la parte superior derecha.

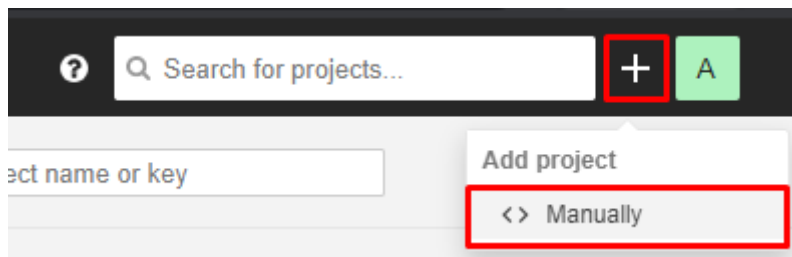


Ilustración 30: SonarQube Add Project

Se indica el nombre del proyecto (*Display name*) y una clave (*Project key*), se genera un token nuevo y se selecciona la tecnología en la que está desarrollada el aplicativo (Java) y la tecnología de compilación (Maven). SonarQube muestra el comando para ejecutar un análisis con la herramienta de construcción de proyectos maven.

```
mvn sonar:sonar \
  -Dsonar.projectKey=WebGoat-Sonar-Jenkins \
  -Dsonar.host.url=http://localhost:9000 \
  -Dsonar.login=0f3fda95df2d2b21121dfe070683829ea818d5c6
```

Ilustración 31: Project WebGoat Sonar Jenkins

6.1.4 Maven

La descarga de maven se realiza desde la página oficial (Apache Maven Project, 2020), a continuación se descomprime el archivo *zip*, se configura la variable de entorno *MAVEN_HOME* y se accede a la terminal introduciendo el comando que se muestra en la imagen:

“mvn –version”

```
C:\Users\José María Acuña>mvn -version
Apache Maven 3.6.3 (cecedd343002696d0abb50b32b541b8a6ba2883f)
Maven home: C:\Hdiv\apache-maven-3.6.3-bin\apache-maven-3.6.3\bin\..
Java version: 1.8.0_241, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk1.8.0_241\jre
Default locale: es_ES, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

Ilustración 32: Apache Maven

6.1.5 Jenkins

Como servidor de automatización se ha optado por Jenkins, ya que es una herramienta de integración continua *open source* que permite automatizar procesos durante el desarrollo del *software* (Heller, 2020). Además, cuenta con una comunidad de desarrolladores muy grande y documentación muy extensa. Jenkins integra procesos en el ciclo de vida de desarrollo de todo tipo, incluidos compilación (*build*), secuencia de comandos de consola, programas por lotes de Windows, *dlls*, *test*, *packages*, despliegue (*deploy*) y análisis estático entre otros.

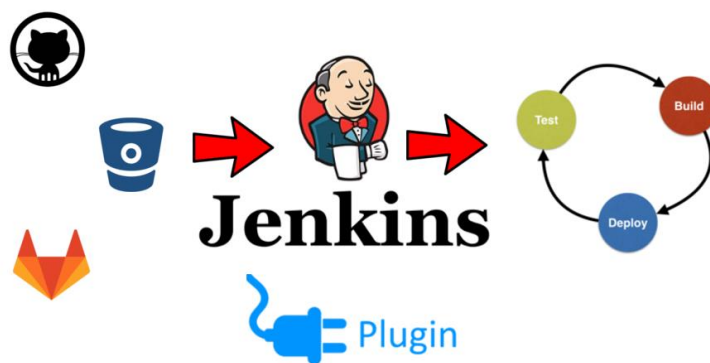


Ilustración 33: Jenkins Plugin

A continuación, se describe el diagrama de flujo genérico de Integración continua con Jenkins:

- En primer lugar, un desarrollador realiza un *commit* en el repositorio de código fuente. Mientras tanto, el servidor de Jenkins comprueba el repositorio a intervalos regulares en busca de cambios.
- Poco después de que se realice un *commit*, el servidor de Jenkins detecta los cambios que se han producido en el repositorio de código fuente. Jenkins aplicará esos cambios y comenzará a preparar una nueva compilación.
- Si la compilación falla, se notificará al equipo implicado enviando un email con información de la *build* en el que se adjunta un fichero de log con todos los detalles.
- Si la compilación tiene éxito, Jenkins despliega la compilación en el servidor de *test*.

- Después de las pruebas, Jenkins notifica al equipo de desarrollo acerca de la compilación y los resultados de los *test*.
- Continuará revisando el repositorio de código fuente en busca de cambios realizados en el código y todo el proceso se seguirá repitiendo.

La siguiente tabla muestra una comparativa antes y después del uso de Jenkins (Ali Stouky, 2020).

Antes y después de Jenkins

Antes de Jenkins	Después de Jenkins
El código fuente completo fue compilado y luego probado. La detección y corrección de errores en caso de fallo de compilación y <i>testing</i> es difícil y lleva mucho tiempo, lo que a su vez ralentiza el proceso de entrega del <i>software</i> .	Cada cambio realizado en el código fuente se compila y se prueba. En lugar de tener que verificar todo el código fuente, los desarrolladores solo necesitan enfocarse en un <i>commit</i> en particular lo que conduce a frecuentes lanzamientos de <i>software</i> nuevos.
Los <i>developers</i> tienen que esperar hasta que se desarrolle el <i>software</i> completo para los resultados de los <i>test</i> .	Los desarrolladores conocen el resultado de los <i>test</i> de cada <i>commit</i> realizado en el código fuente durante la ejecución.
Todo el proceso es manual lo que aumenta el riesgo de errores frecuentes.	Solo necesita realizar cambios en el código fuente y Jenkins automatizará el resto del proceso por usted.

Para su instalación, en primer lugar, se ha descargado el instalador desde los repositorios oficiales (Jenkins, n.d.).

El sistema operativo dónde se va a realizar el desarrollo de este laboratorio es un Windows 10.

Tras la instalación se puede comprobar que el servicio está ejecutándose en el puerto 8080.

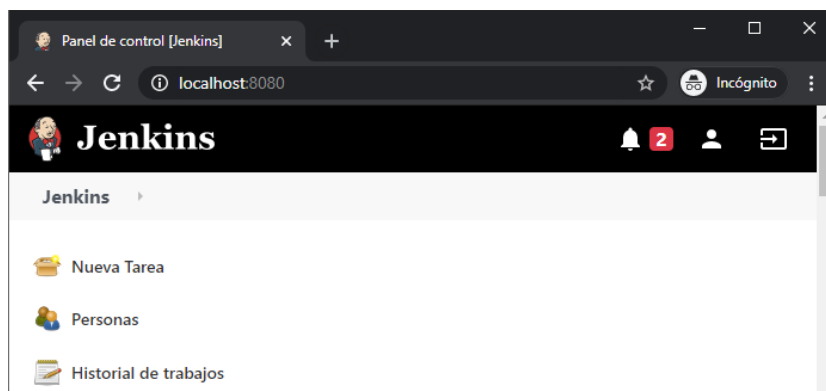


Ilustración 34: Jenkins running localhost:8080

El siguiente paso es la instalación de los *plugins* que se van a necesitar para el laboratorio, entre los que se encuentran GitHub, Sonnar Scanner, Dependency-Check, Docker y OWASP ZAP.

System Configuration



Configurar el Sistema
Configurar variables globales y rutas.



Global Tool Configuration
Configure tools, their locations and automatic installers.



Administrar Plugins
Añadir, borrar, desactivar y activar plugins que extienden la funcionalidad de Jenkins.

Ilustración 35: Jenkins Plugins Administration

<input checked="" type="checkbox"/>	GitHub plugin This plugin integrates GitHub to Jenkins.	1.31.0
<input checked="" type="checkbox"/>	SonarQube Scanner for Jenkins This plugin allows an easy integration of SonarQube , the open source platform for Continuous Inspection of code quality.	2.12
<input checked="" type="checkbox"/>	OWASP Dependency-Check Plugin This plug-in can independently execute a Dependency-Check analysis and visualize results. Dependency-Check is a utility that identifies project dependencies and checks if there are any known, publicly disclosed, vulnerabilities.	5.1.1
<input checked="" type="checkbox"/>	Docker plugin This plugin integrates Jenkins with Docker	1.2.1

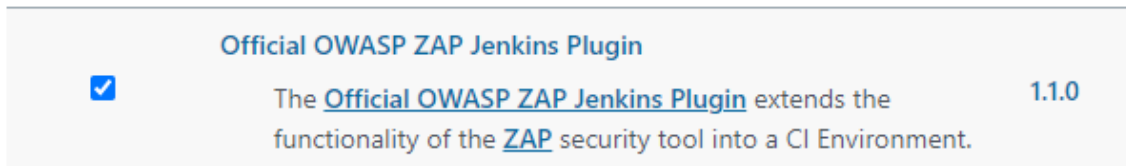


Ilustración 36: Jenkins Plugins

Se continúa con la configuración del servidor de SonarQube en Jenkins. Hay que especificar la ruta dónde se encuentra el servidor.

SonarQube servers

Ilustración 37: Jenkins - SonarQube Server

Del mismo modo, desde el apartado de “*Global Tool Configuration*” se indica dónde se encuentra la ruta de instalación de sonar-scanner a través de la variable “*SONAR_RUNNER_HOME*”.

SonarQube Scanner

Ilustración 38: Jenkins – SonarQube Scanner Configuration

También se especifica la instalación de Dependency-Check. En este caso, se selecciona la última versión disponible que facilita el despliegue y se completa el nombre de la instalación.

Dependency-Check

Ilustración 39: Jenkins – Dependency-Check Install

Ahora se creará una tarea o *job* que va a permitir descargar el código del aplicativo desde el repositorio GitHub, construir la aplicación (*build*) y analizar el código a nivel de seguridad para encontrar potenciales debilidades de seguridad y la calidad de código para mejorar aspectos como la operatividad y el entendimiento de este.

Para ello se accede a la columna izquierda del panel de Jenkins y se hace ‘click’ en la opción “Nueva tarea”, se indica el nombre del *job* “WebGoat”, se añade una descripción del proyecto y se elige la opción “Crear un proyecto de estilo libre”.



Todo	+				
S	W	Nombre ↓	Último Éxito	Último Fallo	Última Duración
		WebGoat	N/D	N/D	N/D

Ilustración 40: Jenkins – Job WebGoat

Una vez creado el proyecto, se accede desde el *link* para visualizarlo.

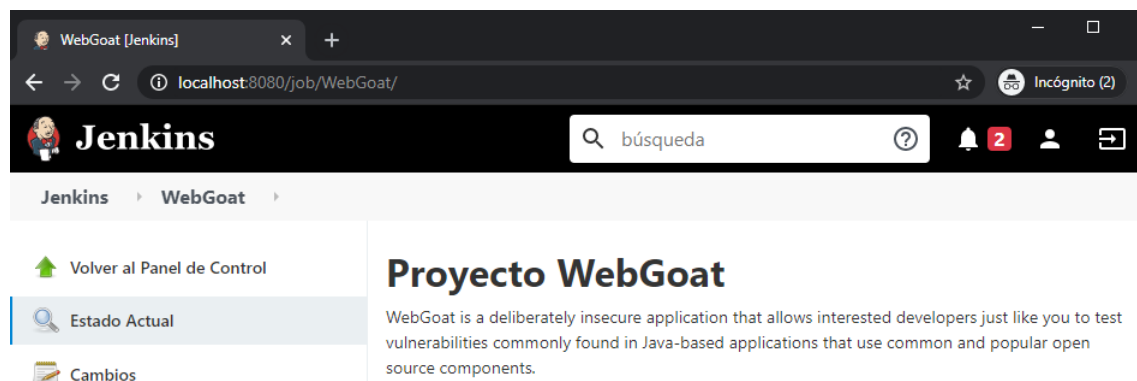
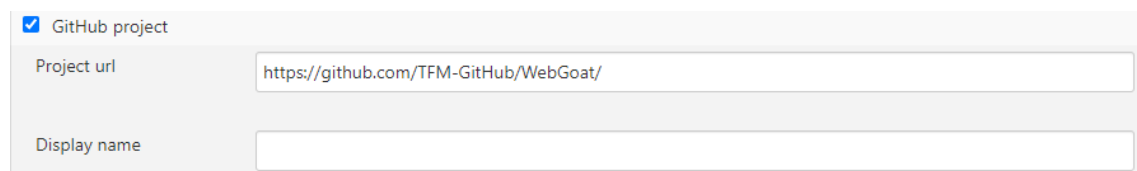


Ilustración 41: Jenkins – Job WebGoat Dashboard

El siguiente paso es configurar el proyecto (opción “configurar”):

- se indica la *url* del proyecto subido a GitHub.



The screenshot shows the 'Configure' page for a Jenkins project. The 'GitHub project' checkbox is checked. The 'Project url' field contains the text 'https://github.com/TFM-GitHub/WebGoat/'. The 'Display name' field is empty.

Ilustración 42: Jenkins – GitHub Project Url

- se indica la *url* del repositorio remoto.

Configurar el origen del código fuente

☐ Ninguno
☒ Git

Repositories

Repository URL

Ilustración 43: Jenkins – GitHub Repository Url

- y las propiedades del análisis para que SonarQube realice el escáner del aplicativo.

Execute SonarQube Scanner

Analysis properties

```
sonar.projectKey=WebGoat-Sonar-Jenkins
sonar.projectName=WebGoat Jenkins
sonar.projectVersion=1.0
sonar.login=e675a9de1eba997b048c4cb9ccab7267aaf9f385
sonar.dependencyCheck.htmlReportPath=target/dependency-check-report.html
sonar.java.binaries=target/classes
```

Ilustración 44: Jenkins – SonarQube Scanner Analysis Configuration

6.1.6 Dependency-Check

El *plugin* Dependency-Check identifica las dependencias del proyecto y comprueba si existen vulnerabilidades conocidas y divulgadas en la NVD o *National Vulnerability Database*.

La herramienta se puede usar como solución al punto número nueve del OWASP Top 10 del año 2017 que identifica este tipo de vulnerabilidades: “A9 – *Using Components with Known Vulnerabilities*” (OWASP Dependency-Check, n.d.).

Este complemento permite visualizar los resultados por separado.

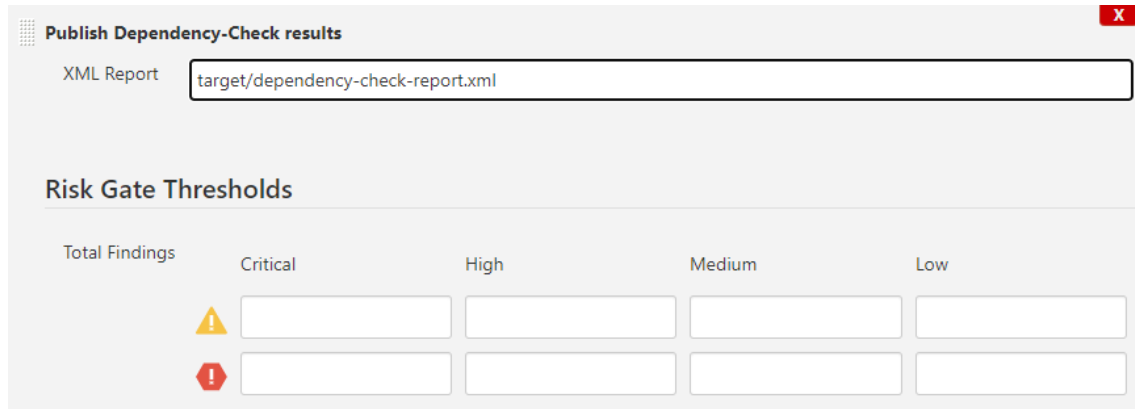
En el *job* de Jenkins se especifica la opción de ejecutar Dependency-Check y la *build* se va a compilar utilizando una llamada a CLI (*Command Line Interface*).

Invoke Dependency-Check

Dependency-Check installation

Ilustración 45: Jenkins – Dependency-Check Installation

El Publisher o Editor actúa independientemente de la configuración de la herramienta o del compilador y se encarga de leer el archivo “*dependency-check-report.xml*” y generar métricas, tendencias, hallazgos y, opcionalmente, definir un estado de advertencias o *warnings* basado en umbrales configurables.



Publish Dependency-Check results					
XML Report					
target/dependency-check-report.xml					
Risk Gate Thresholds					
Total Findings	Critical	High	Medium	Low	
Warning					
Error					

Ilustración 46: Jenkins – Dependency-Check Publish

6.1.7 Docker

El desarrollo de aplicaciones en la actualidad contempla múltiples lenguajes, *frameworks*, arquitecturas e interfaces diferentes generando una complejidad bastante alta en el desarrollo de aplicativos.

Docker simplifica y acelera el flujo de trabajo, permitiendo a los desarrolladores utilizar las últimas herramientas que van apareciendo en el mercado, *stacks* de aplicaciones y entornos de despliegue para cada proyecto.

En definitiva, la tecnología Docker se ha convertido en el estándar de la industria para contenedores, entendidos como un espacio virtualizado de *software* que permite a los desarrolladores aislar las aplicaciones de su entorno, de tal forma que una aplicación con un *stack* tecnológico muy concreto se pueda ejecutar o desplegar en contextos de trabajo muy distintos (Docker Uses Cases, n.d.).

En el proyecto que se viene desarrollando en este trabajo fin de máster se va a utilizar Docker para desplegar la aplicación WebGoat en un contenedor, a efectos de poder usarla en cualquier otro sistema cuyo único requisito es que tenga instalado y corriendo el servicio Docker.

Para este proyecto se ha usado Docker Desktop para Windows, ya que representa la versión *community* de Docker para Microsoft Windows (Docker, n.d.).

La instalación de Docker Desktop incluye Docker Engine, Docker CLI client, Docker Compose, Notary, Kubernetes y Credential Helper.

Una vez ejecutado el instalador Docker Desktop Installer.exe, se procede a iniciar el *software* buscando “Docker” desde los resultados de la búsqueda de Windows.

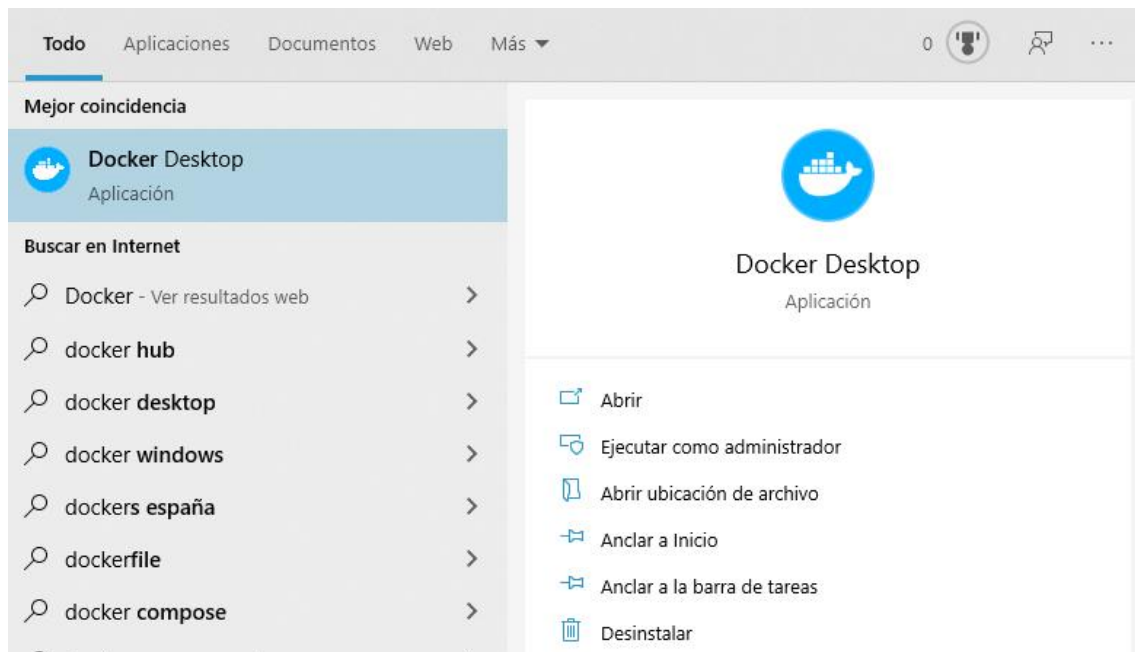


Ilustración 47: Docker Desktop

Y se puede observar que se ha añadido un icono representativo de Docker en la barra de tareas de Windows indicando que el proceso de instalación se ha realizado de forma correcta.



Ilustración 48: Docker Desktop is running

Cuando se completa la instalación, Docker Desktop ofrece un tutorial que incluye un ejercicio sencillo para crear una imagen de ejemplo, ejecutarla como un contenedor, añadir (*push*) y guardar la imagen en Docker Hub.

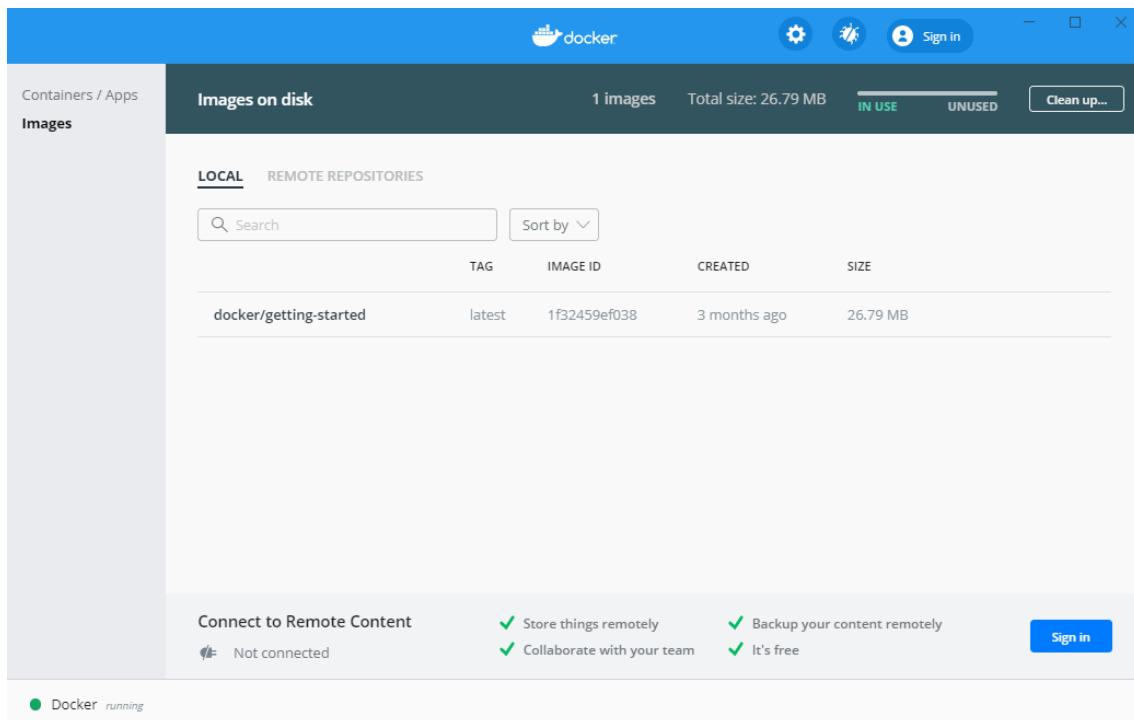


Ilustración 49: Docker Desktop Dashboard

El objetivo principal de los contenedores es incluir todo lo necesario para que una aplicación pueda ejecutarse en cualquier plataforma sin tener que cambiar nada.

Un contenedor sólo es posible crearlo a partir de una imagen. Una imagen, a grandes rasgos, es una plantilla que permite crear contenedores. Incluye el diseño y la configuración necesaria para poder crear contenedores concretos. En Jenkins, se tiene que acceder al proyecto WebGoat y ejecutar los comandos Docker que se observan en la ilustración 49 para la creación de la imagen y el contenedor donde se va a desplegar la aplicación que se va a ejecutar en el navegador.

Se especifica la ruta y el nombre del archivo de texto *Dockerfile* que contiene las instrucciones necesarias para crear una imagen.

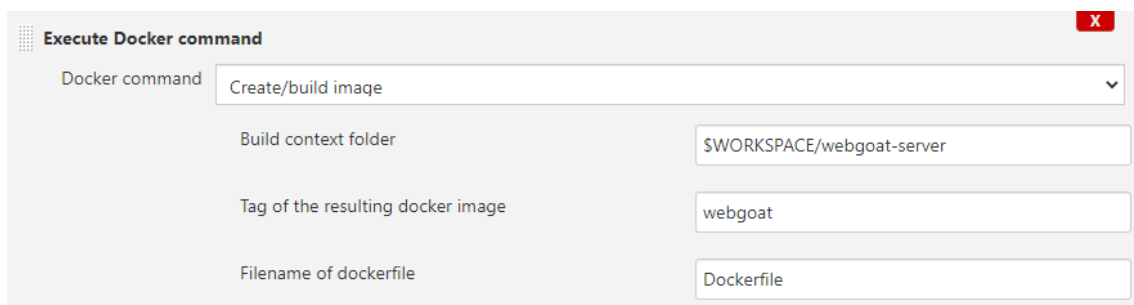


Ilustración 50: Jenkins – Docker Create Image

Este fichero viene por defecto en el proyecto oficial WebGoat descargado de GitHub.

```
1 FROM openjdk:11.0.1-jre-slim-stretch
2
3 ARG webgoat_version=v8.1.0
4
5 RUN \
6     apt-get update && apt-get install && \
7     useradd --home-dir /home/webgoat --create-home -U webgoat
8
9 USER webgoat
10 RUN cd /home/webgoat/; mkdir -p .webgoat-${webgoat_version}
11 COPY target/webgoat-server-${webgoat_version}.jar /home/webgoat/webgoat.jar
12
13 EXPOSE 8080
14
15 WORKDIR /home/webgoat
16 ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "/home/webgoat/webgoat.jar"]
17 CMD ["--server.port=8080", "--server.address=0.0.0.0"]
```

Ilustración 51: Dockerfile

Se añade el comando para crear un contenedor a partir del nombre de la imagen, se le da un nombre y se especifica el puerto dónde se va a desplegar, en este caso el puerto 9999 ya que por defecto utiliza el 8080 que está usando la aplicación Jenkins.

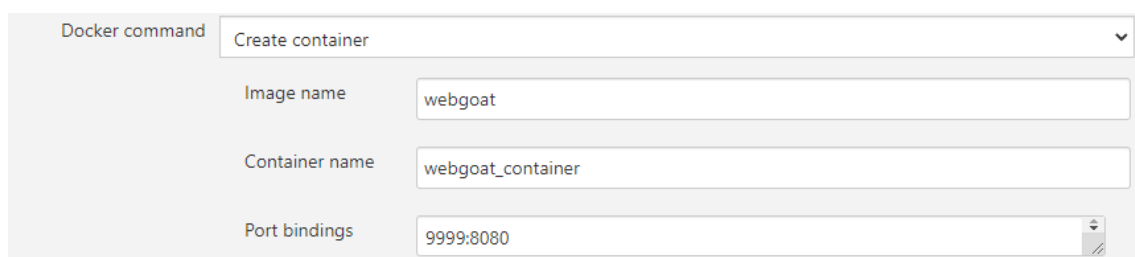
The screenshot shows the 'Create container' form in Jenkins. The 'Docker command' dropdown is set to 'Create container'. The 'Image name' field contains 'webgoat'. The 'Container name' field contains 'webgoat_container'. The 'Port bindings' field contains '9999:8080'.

Ilustración 52: Jenkins – Docker Create Container

Por último, se indica que inicie el contenedor una vez creado.

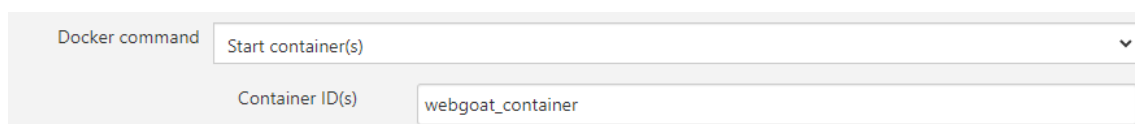
The screenshot shows the 'Start container(s)' form in Jenkins. The 'Docker command' dropdown is set to 'Start container(s)'. The 'Container ID(s)' field contains 'webgoat_container'.

Ilustración 53: Jenkins – Docker Start Container

6.1.8 MailDev

Esta herramienta permite su uso con diferentes tecnologías como node, Django, Rails, Drupal, entre otros. Algunas de las características más reseñables son:

- Permite el envío de emails en formato *HTML*, formato *Plain Text* y visualizar las cabeceras de los mensajes.
- Se pueden adjuntar archivos al correo electrónico.
- Proporciona una interfaz de línea de comandos para configurar los

puertos del servicio web y del servidor SMTP.

En este trabajo fin de máster se va a utilizar usando la imagen “*maildev*” existente en Docker Hub (maildev, n.d.).

Para ello, se accede a la consola para crear la imagen “*maildev/maildev*” ejecutando el comando:

docker pull maildev/maildev

A continuación, se crea el contenedor “*maildev*” con el cliente web corriendo en el puerto 1080 y el servidor SMTP en el puerto 25:

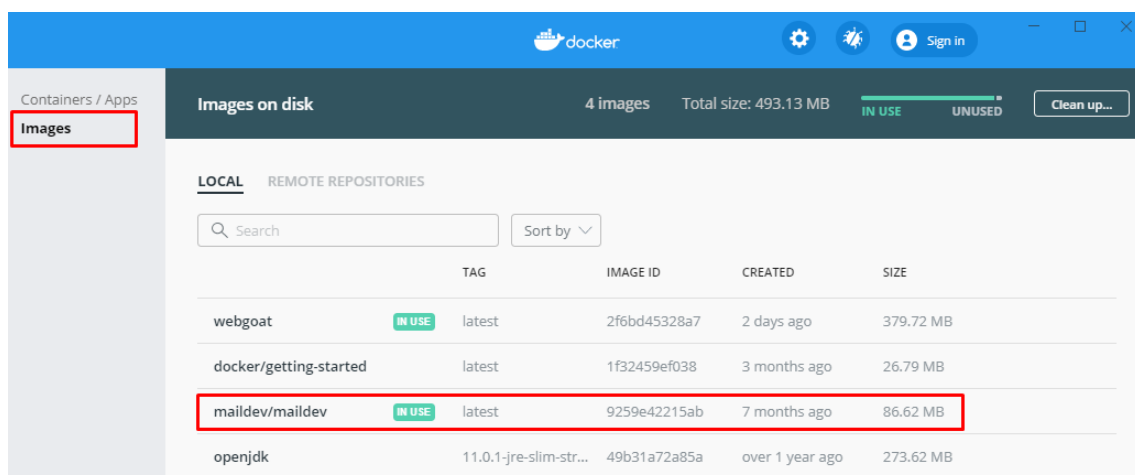
docker run -p 1080:80 -p 25:25 --name maildev maildev/maildev

```
C:\WINDOWS\system32>docker pull maildev/maildev
Using default tag: latest
latest: Pulling from maildev/maildev
e6b0cf9c0882: Pull complete
93f9cf0467ca: Pull complete
a564402f98da: Pull complete
b68680f1d28f: Pull complete
d83a90929b44: Pull complete
5bb08f80fc87: Pull complete
021ced319bab: Pull complete
7a42c2dca0ef: Pull complete
Digest: sha256:9ae76db9e72ad3c41a34ffcc327bbd3525849a161d257888f41a8dc4262ec73f
Status: Downloaded newer image for maildev/maildev:latest
docker.io/maildev/maildev:latest

C:\WINDOWS\system32>docker run -p 1080:80 -p 25:25 --name maildev maildev/maildev
MailDev webapp running at http://0.0.0.0:80
MailDev SMTP Server running at 0.0.0.0:25
```

Ilustración 54: Docker Pull MailDev

Desde Docker Desktop se observa que se han creado la imagen y el contenedor correctamente.



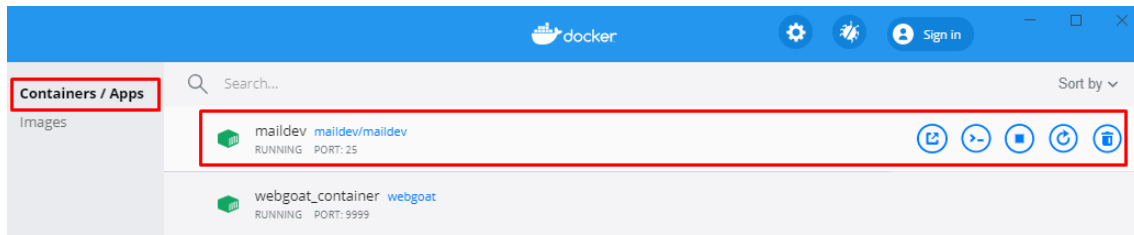


Ilustración 55: Docker – Image & Container MailDev

Se dispone ya de la interfaz de usuario de MailDev corriendo en el puerto 1080.

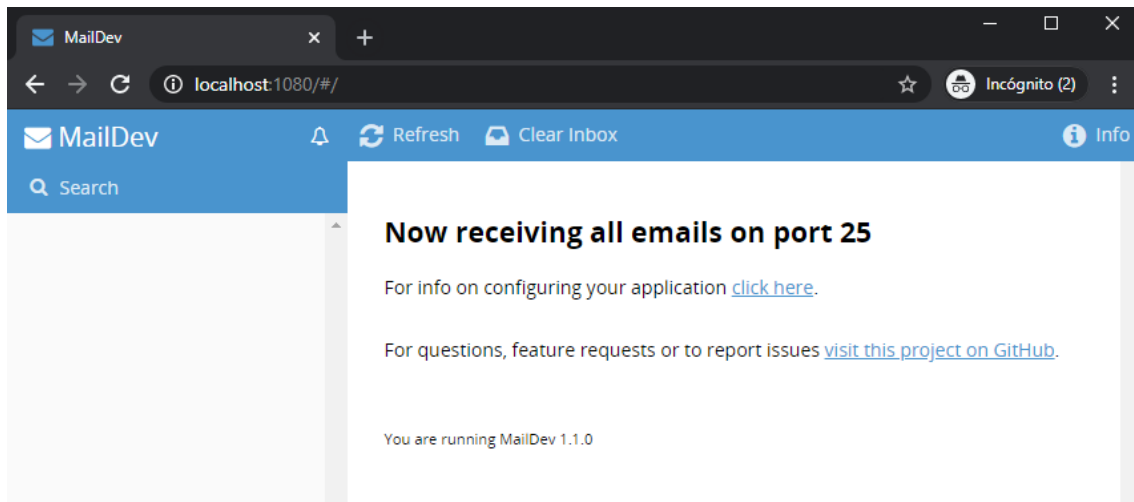


Ilustración 56: MailDev – http 1080

A continuación, se accede al panel de Administración de Jenkins entrando en la opción *Configurar el Sistema*. En la sección "*Extended E-mail Notification*" se añade el servidor *SMTP* local, el puerto 25 correspondiente al estándar de *SMTP*, el *Content-Type* del documento que, en este caso, será *HTML*, la codificación se definirá como UTF-8 y el asunto del correo.

En el campo *Default Content* se añaden las etiquetas *html* para dar formato al email que se va a enviar al equipo de desarrollo.

Extended E-mail Notification

SMTP Server	<input type="text" value="localhost"/>
SMTP Port	<input type="text" value="25"/>
Charset	<input type="text" value="UTF-8"/>
Default Content Type	<input type="text" value="HTML (text/html)"/>
Excluded Recipients	<input type="text"/>
Default Subject	<input type="text" value="\$PROJECT_NAME - Build # \$BUILD_NUMBER - \$BUILD_STATUS!"/>
Default Content	<pre><div class="container"> <div class="row text-center"> <div class="col-sm-6 col-sm-offset-3">

 <script> if('\$BUILD_STATUS' == 'Successful' '\$BUILD_STATUS' == 'Fixed'){ document.write("<h2 style='color:#0fad00'>EXECUTION \$BUILD_STATUS</h2>\n "); }else{</pre>

Ilustración 57: Jenkins – Extended E-mail Notification

Para realizar una prueba de envío de correo electrónico a la dirección devops@jenkins.com se pulsa el botón “Probar configuración”

Notificación por correo electrónico

Servidor de correo saliente (SMTP)	<input type="text" value="localhost"/>
Sufijo de email por defecto	<input type="text" value="@jenkins.com"/>
<input checked="" type="checkbox"/> Probar la configuración enviando un correo de prueba	
Dirección para el correo de prueba	<input type="text" value="devops@jenkins.com"/>
Email was successfully sent	
<input type="button" value="Probar configuración"/>	

Ilustración 58: Jenkins – Extended Test Mail

El email se ha enviado de forma satisfactoria y se puede visualizar accediendo a la interfaz web.

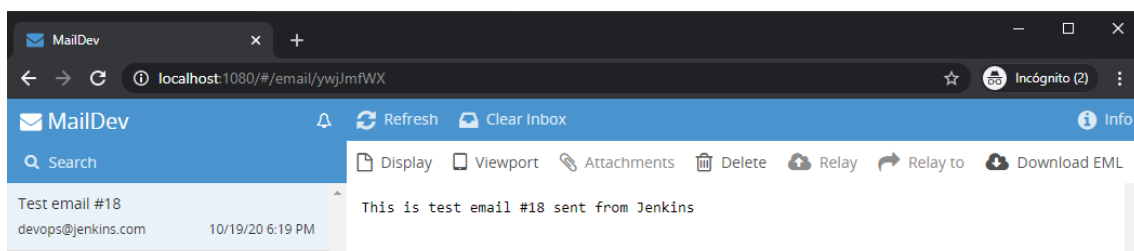
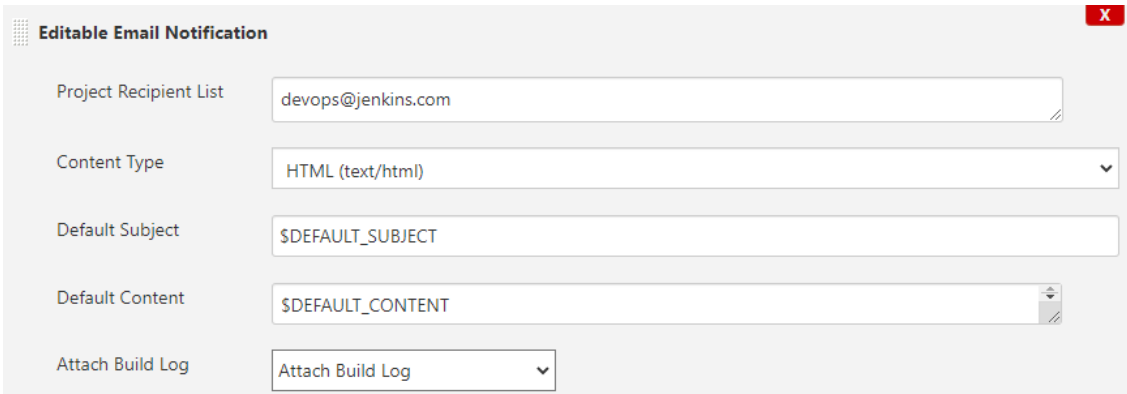


Ilustración 59: MailDev – Test Mail

Después desde el proyecto WebGoat se configura la sección “*Editable Email Notification*” con los siguientes parámetros:



Editable Email Notification	
Project Recipient List	devops@jenkins.com
Content Type	HTML (text/html)
Default Subject	\$DEFAULT_SUBJECT
Default Content	\$DEFAULT_CONTENT
Attach Build Log	Attach Build Log

Ilustración 60: Jenkins – Editable Email Notification

6.1.9 OWASP ZAP

OWASP Zed Attack Proxy (ZAP) es uno de los escáneres de aplicaciones web más utilizado del mundo, mantenido por un equipo de voluntarios a nivel internacional. Se trata de una herramienta de pruebas de seguridad de aplicaciones web que ayuda a evaluar y detectar vulnerabilidades. Está basada en Java y dispone de una interfaz gráfica intuitiva, lo que permite a los *pentesters* de aplicaciones web realizar *fuzzing*, entendido como el conjunto de pruebas automáticas introduciendo datos aleatorios para descubrir errores en la aplicación, *scripting* para modificar funcionalidades y ampliar las características de la herramienta a través de la interfaz y *spidering* para explorar toda la web de forma automática; además de configurar un proxy para atacar aplicaciones web (ZAP Add-ons, n.d.).

Las versiones de Windows y Linux requieren Java 8 o superior para ejecutarse. Para la descarga de la herramienta se accede a la página oficial de zaproxy (OWASP ZAP, n.d.) y se realiza la instalación en el puerto 9999 del proxy dónde corre la aplicación desplegada en Docker.



Ilustración 61: OWASP ZAP proxy port

En la configuración del sistema de Jenkins, se establece el puerto dónde corre la aplicación, en este caso, como se ha visto, es el puerto 9999 y el método de instalación, gestión y propiedades de la sesión, modo de ataque y nombre del fichero para generar el reporte.

ZAP

Default Host	<input type="text" value="localhost"/>
Default Port	<input type="text" value="9999"/>

Installation Method

- ☐ Custom Tools Installation
- ☒ System Installed: ZAP Installation Directory

Environment Variable

Session Management

- ☐ Load Session
- ☒ Persist Session

Filename



The image shows a configuration window for OWASP ZAP in Jenkins. It is divided into two main sections: 'Session Properties' and 'Attack Mode'. In the 'Session Properties' section, there are two input fields: 'Context Name' with the value 'ZAPWebGoat' and 'Include in Context' with the value 'http://localhost:9999/WebGoat/'. In the 'Attack Mode' section, there is one input field: 'Starting Point' with the value 'http://localhost:9999/WebGoat/'.

Session Properties	
Context Name	ZAPWebGoat
Include in Context	http://localhost:9999/WebGoat/

Attack Mode	
Starting Point	http://localhost:9999/WebGoat/

Ilustración 62: Jenkins – OWASP ZAP Configuration

6.1.10 Jira

Jira se presentó en el año 2002 como una plataforma de seguimiento de errores para los desarrolladores de *software* pero, hoy en día, se ha convertido en un *software* muy completo para la gestión de proyectos.

Entre sus funcionalidades cabe destacar:

- Gestión de proyectos Agile (scrum, kanban).
- *Bug tracking* para el seguimiento de errores de *software* a lo largo del flujo de trabajo.
- Gestión de contenidos con la implementación de plantillas personalizadas.
- Lanzamiento de campañas y productos de marketing.

Proporciona un conjunto de funciones específicas como la integración con herramientas de desarrollo (GitHub, GitLab, Fisheye, Bitbucket), *dashboards* y centro de lanzamiento para versiones de *software*, entre otros.

El plan *free* admite hasta 10 usuarios o 3 agentes, incluye 2 GB de almacenamiento y ofrece soporte de la comunidad.

Se va a aprovechar la funcionalidad de seguimiento de tareas para realizar una notificación a un proyecto Jira creado para tal uso, una vez ha finalizado la compilación del *job* de Jenkins.

Para ello, hay que registrarse en la plataforma Atlassian (Jira Software, s.f.) seleccionar el producto Jira Software y elegir un nombre para el sitio web. El nombre escogido en este proyecto es **tfm-devsecops**



Comencemos

Asigna un nombre a tu sitio

Elige un nombre conocido, como el del equipo o la empresa

✓

Ilustración 63: Web Site Atlassian

Una vez realizado el registro se accede a la aplicación a través de la *url* local:

<https://tfm-devsecops.atlassian.net>

El siguiente paso es crear tantos proyectos como sea necesario.

Crear proyecto

Nombre *

Clave *



Plantilla



Kanban

Visualiza tu proyecto y llévalo hacia delante usando tarjetas sencillas en un tablero lleno de posibilidades.

[Quiero saber más](#)

[Cambiar plantilla](#)

Crear

Ilustración 64: New Project Atlassian

A partir de este proyecto, se accede a la opción Tablero y se empiezan a crear tareas o incidencias.

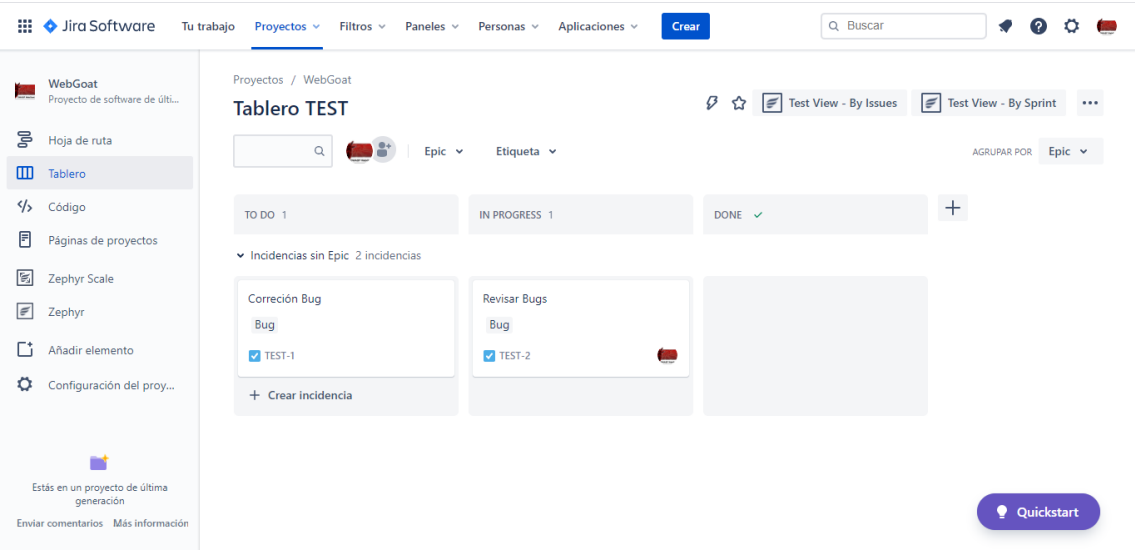


Ilustración 65: Board Jira Software

La opción *Default Dashboard* ofrece una visión global de los proyectos y tareas construidos.

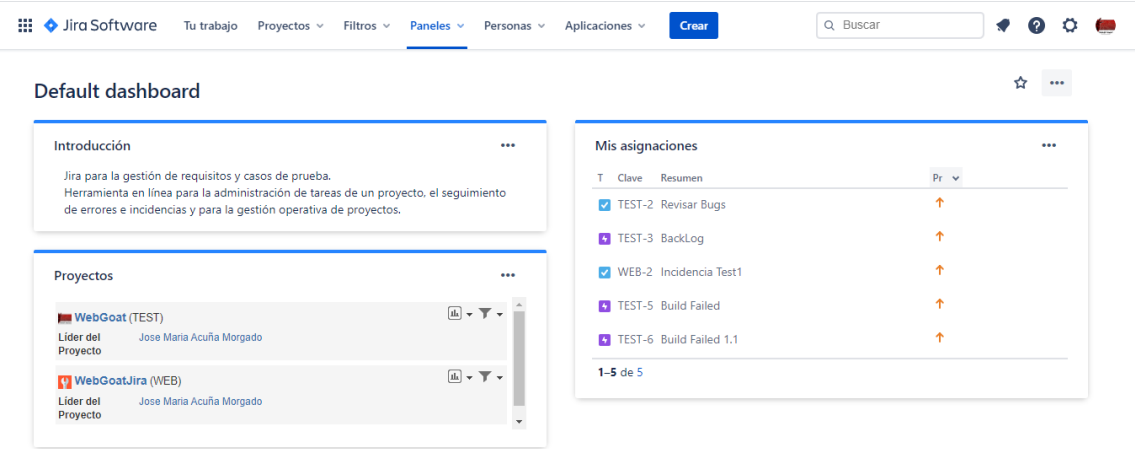


Ilustración 66: Default Dashboard Jira

Del mismo modo, la plataforma muestra un panel con la actividad que se ha ido realizando en los proyectos como se puede observar en la siguiente imagen:

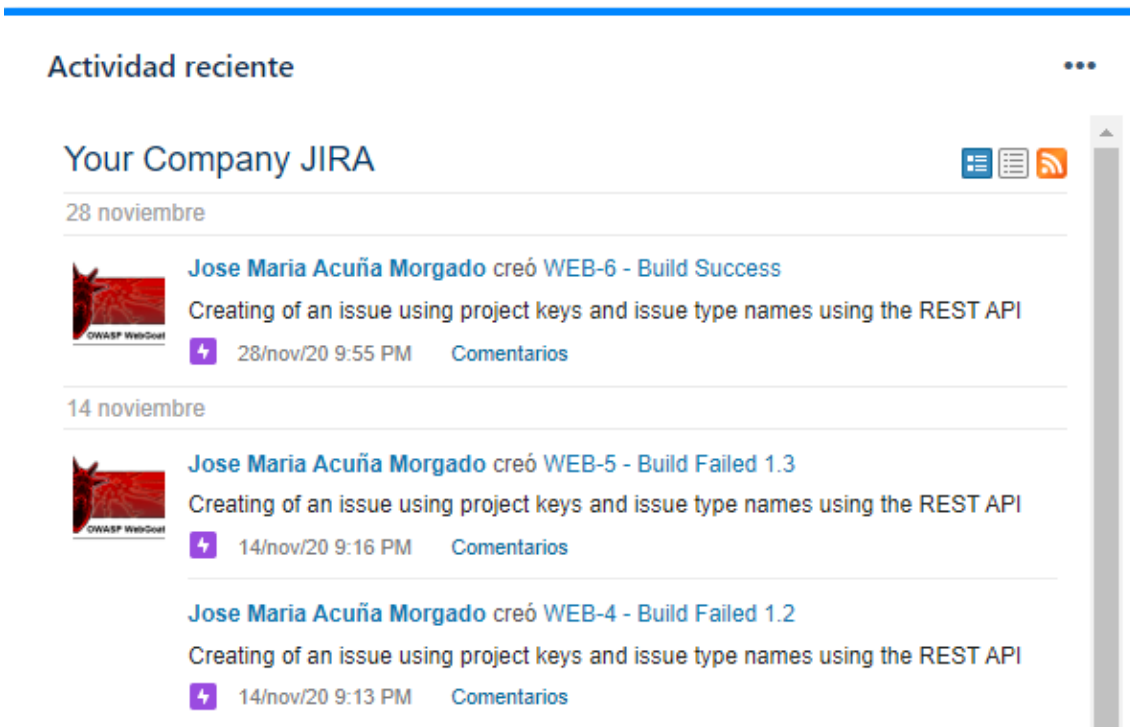


Ilustración 67: Recent Activity Jira

Para la integración con Jenkins, Jira ofrece una API REST consumible mediante curl. Curl es una herramienta *open source* de línea de comandos para la transferencia de datos con *URLs*.

La página oficial para desarrolladores (Jira REST API examples, n.d.) muestra numerosos ejemplos de uso de la API REST de Jira.

Para añadir una notificación a un tablero de Jira se requiere especificar una serie de metadatos para invocar mediante el comando curl. Entre ellos están la URL de la *issue*, la clave del proyecto, el tipo de *issue*, el ID de usuario o dirección de correo electrónico y un token de autenticación. Dicho token se puede crear desde la plataforma de seguridad para la gestión de tokens (Security - Api Tokens Jira, n.d.).

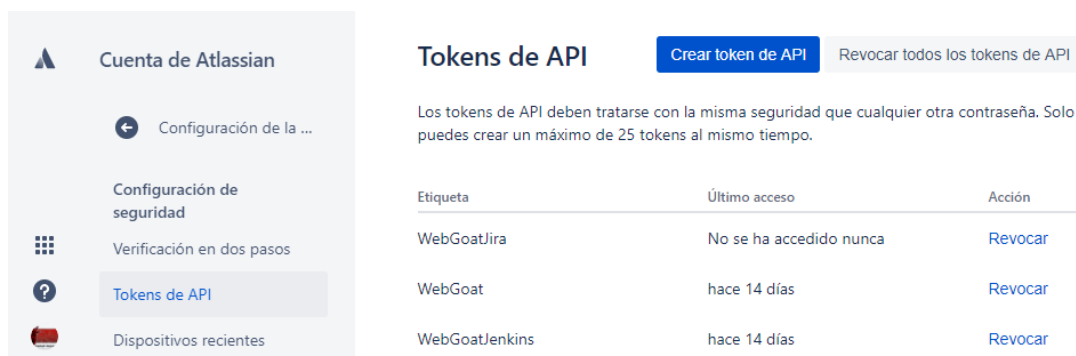


Ilustración 68: Token Jira

Acciones para ejecutar después.

Ejecutar otros proyectos

Proyectos a ejecutar

Jira

☒ Trigger only if build is stable

A continuación, se crea un nuevo *job* en Jenkins con el nombre ‘Jira’ (éste será hijo del *job* principal WebGoat) y en la opción “Ejecutar un comando de Windows” se detalla la siguiente información:

Ilustración 69: Run Windows Command

```
add_issue.txt
1 {
2   "fields": {
3     "project":
4     {
5       "key": "WEB"
6     },
7     "summary": "Build Success",
8     "description": "Creating of an issue using project keys and issue type names using the REST API",
9     "issuetype": {
10      "name": "Epic"
11    }
12  }
13 }
```

44

7. Resultados obtenidos tras la construcción del proyecto

Tras realizar una *build* del *job* de Jenkins, se observan a través de la consola del *job*, los logs de todas las fases de la *build* y que se han ido describiendo en puntos anteriores del documento:

7.1 Jenkins y GitHub

El sistema se conecta al repositorio remoto Git para comprobar si hay cambios y realiza un *checkout* en el repositorio local.

```
Running as SYSTEM
Ejecutando en el espacio de trabajo C:\Users\José María Acuña\.jenkins\workspace\WebGoat
The recommended git tool is: NONE
No credentials specified
> git.exe rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git.exe config remote.origin.url https://github.com/TFM-GitHub/WebGoat.git # timeout=10
Fetching upstream changes from https://github.com/TFM-GitHub/WebGoat.git
> git.exe --version # timeout=10
> git --version # 'git version 2.28.0.windows.1'
> git.exe fetch --tags --force --progress -- https://github.com/TFM-GitHub/WebGoat.git
+refs/heads/*:refs/remotes/origin/* # timeout=10
> git.exe rev-parse "refs/remotes/origin/master^{commit}" # timeout=10
> git.exe rev-parse "refs/remotes/origin/origin/master^{commit}" # timeout=10
Checking out Revision 69b0a2d795b227f74bf7bfdbf85d88cb89dbd41e (refs/remotes/origin/master)
> git.exe config core.sparsecheckout # timeout=10
> git.exe checkout -f 69b0a2d795b227f74bf7bfdbf85d88cb89dbd41e # timeout=10
Commit message: "Sever Port: 8080"
> git.exe rev-list --no-walk 69b0a2d795b227f74bf7bfdbf85d88cb89dbd41e # timeout=10
```

Ilustración 71: Jenkins – GitHub Output

7.2 Jenkins y SonarQube

El siguiente paso de ejecución del *job* es el análisis de código estático a través de los parámetros de configuración de SonarQube.

```
[WebGoat] $ C:\TFM-Deloitte\sonar-scanner-cli-4.4.0.2170-windows\sonar-scanner-4.4.0.2170-
windows\bin\sonar-scanner.bat -Dsonar.host.url=http://localhost:9000 -Dsonar.projectKey=WebGoat-
Sonar-Jenkins "-Dsonar.projectName=WebGoat Jenkins" -
Dsonar.login=e675a9de1eba997b048c4cb9ccab7267aaf9f385 -Dsonar.projectVersion=1.0 -
Dsonar.dependencyCheck.htmlReportPath=target/dependency-check-report.html -
Dsonar.java.binaries=target/classes "-Dsonar.projectBaseDir=C:\Users\José María
Acuña\.jenkins\workspace\WebGoat"
INFO: Scanner configuration file: C:\TFM-Deloitte\sonar-scanner-cli-4.4.0.2170-windows\sonar-
scanner-4.4.0.2170-windows\bin\..\conf\sonar-scanner.properties
INFO: Project root configuration file: NONE
INFO: SonarScanner 4.4.0.2170
INFO: Java 11.0.3 AdoptOpenJDK (64-bit)
INFO: Windows 10 10.0 amd64
INFO: User cache: C:\Users\José María Acuña\.sonar\cache
INFO: Scanner configuration file: C:\TFM-Deloitte\sonar-scanner-cli-4.4.0.2170-windows\sonar-
scanner-4.4.0.2170-windows\bin\..\conf\sonar-scanner.properties
INFO: Project root configuration file: NONE
INFO: Analyzing on SonarQube server 8.4.2
INFO: Default locale: "es_ES", source code encoding: "windows-1252" (analysis is platform
dependent)
```

Ilustración 72: Jenkins – SonarQube Output

El análisis del proyecto se ha producido de forma satisfactoria y se han generado los reportes en la ruta que indica el *job*, en el puerto 9000 que instala por defecto el servidor de SonarQube.

```
INFO: Analysis report generated in 712ms, dir size=27 MB
INFO: Analysis report compressed in 18387ms, zip size=5 MB
INFO: Analysis report uploaded in 175ms
INFO: ANALYSIS SUCCESSFUL, you can browse http://localhost:9000/dashboard?id=WebGoat-Sonar-Jenkins
INFO: Note that you will be able to access the updated dashboard once the server has processed the submitted
analysis report
INFO: More about the report processing at http://localhost:9000/api/ce/task?id=AXU12Y9k-J8SLr9UvCjn
INFO: Analysis total time: 1:51.445 s
INFO: -----
INFO: EXECUTION SUCCESS
INFO: -----
INFO: Total time: 1:54.517s
INFO: Final Memory: 19M/74M
INFO: -----
```

Ilustración 73: Jenkins - SonarQube Execution Success

El *dashboard* de SonarQube ofrece los resultados del análisis que se acaba de realizar.

The screenshot shows the SonarQube web interface. At the top, there's a navigation bar with 'Projects', 'Issues', 'Rules', 'Quality Profiles', and 'Quality Gates'. Below this, a filter section on the left shows 'Quality Gate' with 'Passed' (1) and 'Failed' (0) counts, and 'Reliability' (0 bugs). The main area displays the 'WebGoat Jenkins' project with a green 'Passed' badge. Below the project name, it says 'Last analysis: October 17, 2020, 11:15 AM'. A row of metrics follows: 465 bugs (red E), 10 vulnerabilities (red E), 0.0% hotspots reviewed (red E), 2.1k code smells (green A), 0.0% coverage (red circle), 20.0% duplications (red circle), and 110k lines of code (blue L). The bottom of the dashboard shows a search bar and a 'Log in' button.

Ilustración 74: SonarQube Analysis Dashboard

El panel aporta información muy útil en cuanto a evidencias como son el número de *bugs*, vulnerabilidades, *security hotspots* que son bloques de código a revisar por el auditor de seguridad para valorar si se trata o no de una vulnerabilidad, *code smells* entendidos como líneas o bloques de código con una baja o mala calidad en su implementación, y líneas de código (SonarQube User Guide, n.d.).

El número de *bugs* es muy elevado lo que denota una mala calidad del *software*. Además, se han detectado 10 vulnerabilidades y 143 *security hotspots*.

La siguiente imagen muestra un panel con información de las evidencias detectadas por el analizador:

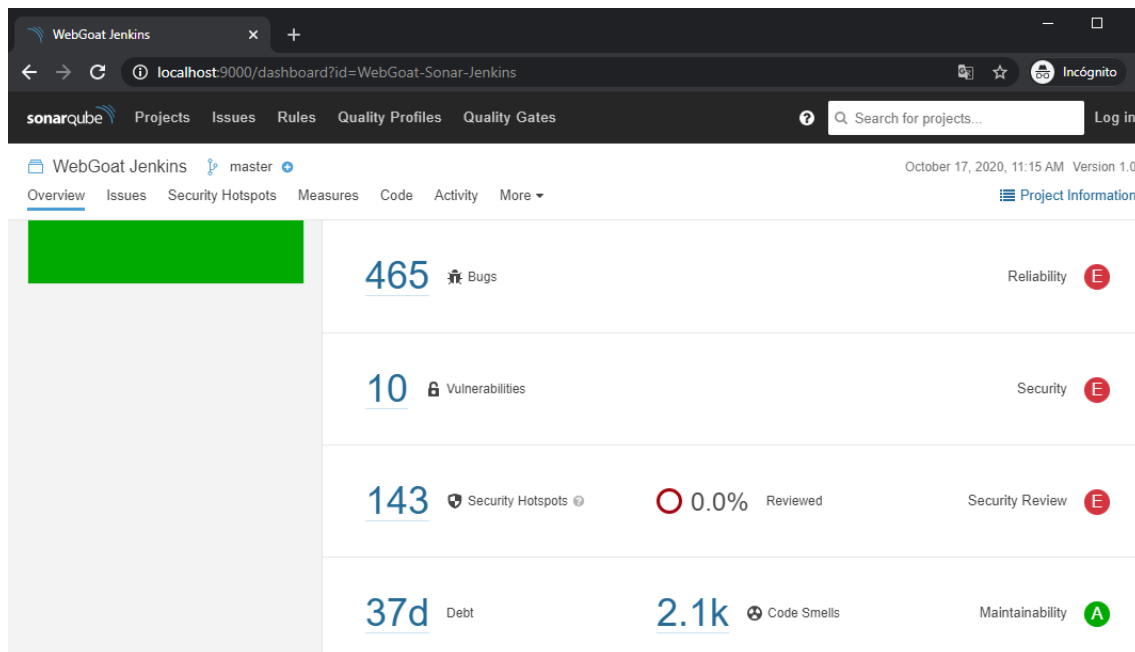


Ilustración 75: SonarQube Analysis Results

Al acceder al detalle de las vulnerabilidades, aparecen 4 de tipo bloqueante. Del mismo modo, en cuando a *security hotspots*, hay 22 evidencias de tipo High que pertenecen a la categoría 2 de *OWASP Top 10 2017, A2 - Broken Authentication* y 9 evidencias High encuadradas en la categoría *A1- Injection*. Además, la herramienta proporciona información útil sobre vulnerabilidades conocidas con sus CVE, información sobre los riesgos que subyacen en la aplicación y facilita posibles remediaciones.

What's the risk?	Are you at risk?	How can you fix it?
<p>Because it is easy to extract strings from an application source code or binary, credentials should not be hard-coded. This is particularly true for applications that are distributed or that are open-source.</p> <p>In the past, it has led to the following vulnerabilities:</p> <ul style="list-style-type: none"> • CVE-2019-13466 • CVE-2018-15389 		

Ilustración 76: SonarQube Risk

Hasta que no se hayan solucionado las evidencias de categoría Bloqueante o Alta, el código de la aplicación no se debería subir al entorno de producción. Además, el equipo de desarrollo debería evaluar los *bugs* y *code smell* para mejorar la calidad del código del aplicativo.

The screenshot shows the SonarQube Security Hotspots interface. The top navigation bar includes 'sonarqube', 'Projects', 'Issues', 'Rules', 'Quality Profiles', and 'Quality Gates'. The main content area is titled 'WebGoat Jenkins' and shows a list of hotspots. The first hotspot is highlighted, showing the message: 'password' detected in this expression, review this potentially hard-coded credential. The status is 'TO REVIEW'. The detailed view on the right shows the category 'Authentication', review priority 'HIGH', and assignee 'Not assigned'. The status is 'To review' with a note: 'This Security Hotspot needs to be reviewed to assess whether the code poses a risk.' The code snippet shown is from 'webgoat-integration-tests/src/test/java/org/owasp/webgoat/GeneralLessonTest.java' and contains a method 'insecureLogin()' with a hard-coded password 'CaptainJack'.

Ilustración 77: SonarQube Security Hotspots

7.3 Jenkins y Dependency-Check

El siguiente paso en la construcción del *job*, es el análisis de dependencias o componentes de terceros que proporciona Dependency-Check.

```

[DependencyCheck] 00:10 INFO: Vulnerability found: angularjs below 1.5.0-rc2
[DependencyCheck] 00:10 INFO: Vulnerability found: angularjs below 1.6.3
[DependencyCheck] 00:10 INFO: Vulnerability found: angularjs below 1.6.3
[DependencyCheck] 00:10 INFO: Vulnerability found: angularjs below 1.6.5
[DependencyCheck] 00:11 INFO: Vulnerability found: jquery below 1.9.0b1
[DependencyCheck] 00:11 INFO: Vulnerability found: jquery below 1.12.0
[DependencyCheck] 00:11 INFO: Vulnerability found: jquery below 3.4.0
[DependencyCheck] 00:11 INFO: Vulnerability found: jquery below 3.5.0
[DependencyCheck] 00:11 INFO: Vulnerability found: jquery below 3.5.0
[DependencyCheck] [INFO] Finished RetireJS Analyzer (11 seconds)
[DependencyCheck] [INFO] Finished Sonatype OSS Index Analyzer (0 seconds)
[DependencyCheck] [INFO] Finished Vulnerability Suppression Analyzer (0 seconds)
[DependencyCheck] [INFO] Finished Dependency Bundling Analyzer (11 seconds)
[DependencyCheck] [INFO] Analysis Complete (61 seconds)
[DependencyCheck] Collecting Dependency-Check artifact
Finished: SUCCESS

```

Ilustración 78: Jenkins - DependencyCheck Output

En el *dashboard* del proyecto WebGoat aparece un gráfico de tendencias de Dependency-Check en el que se muestra el número de vulnerabilidades por nivel de riesgo y criticidad: Críticas, Altas, Medias y Bajas.

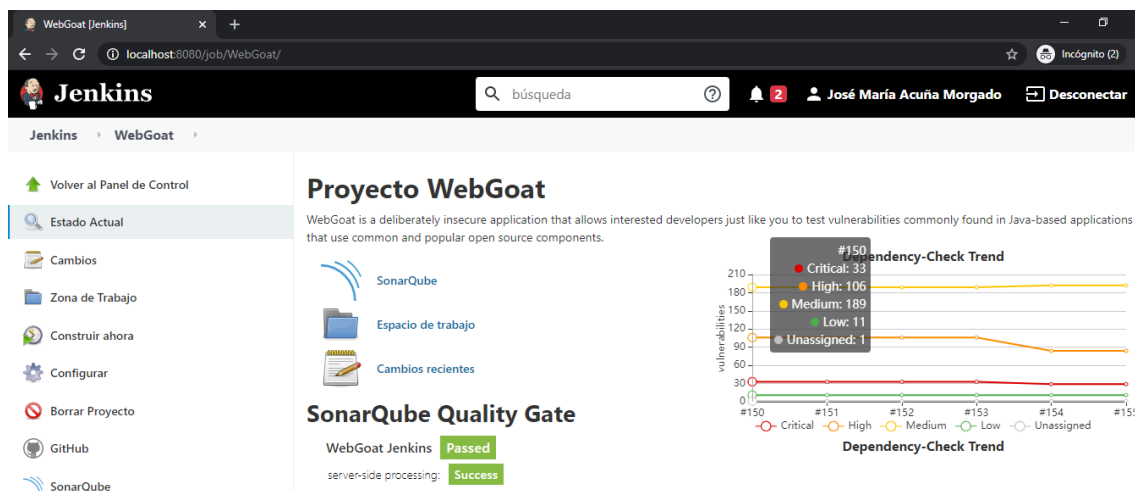


Ilustración 79: Jenkins - DependencyCheck Trend

Además, desde una *build* en particular, aparecen numeradas, se puede visualizar la tabla de resultados de las vulnerabilidades desde la opción del menú.

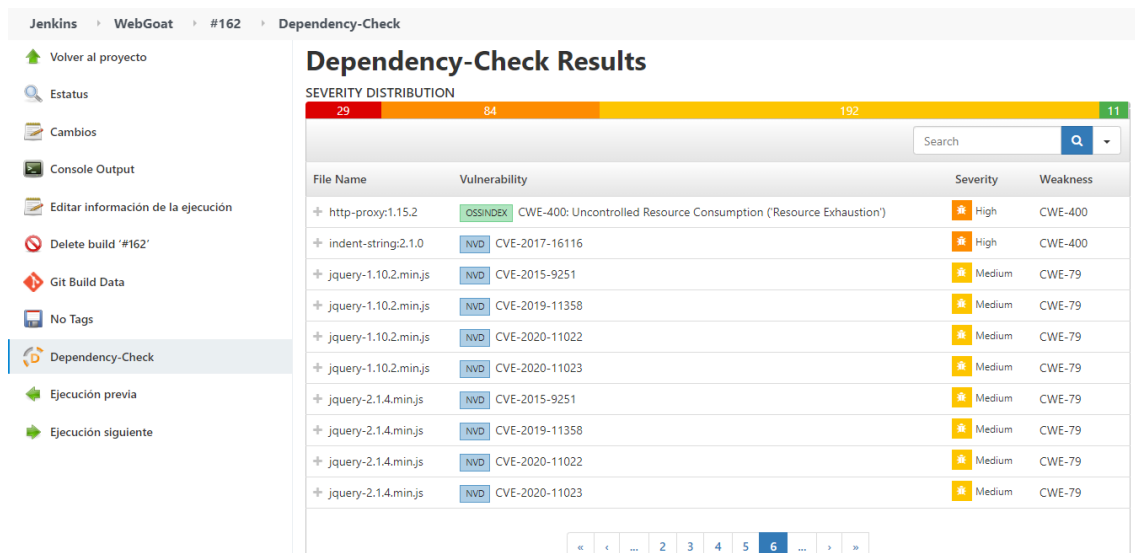


Ilustración 80: Jenkins - DependencyCheck Results

En SonarQube, también se ha añadido el *plugin* Dependency-Check y Jenkins vuelca los resultados del análisis desde la pestaña *More* del panel.

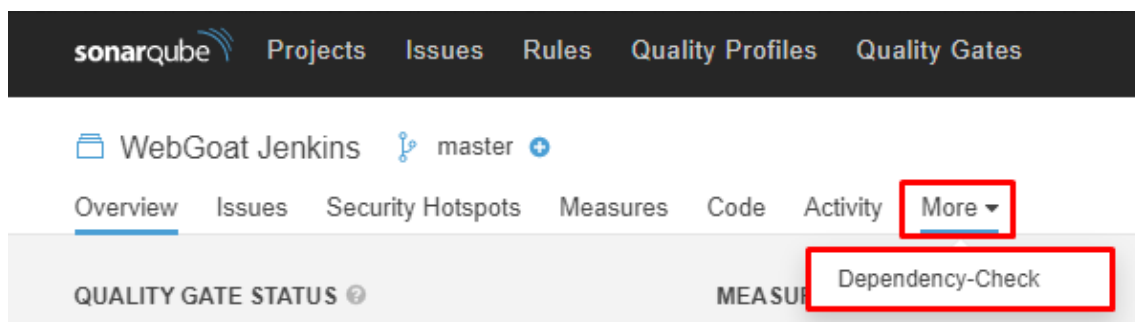



Ilustración 81: SonarQube - DependencyCheck Tab More

Accediendo a la opción del menú aparece información sobre el escáner como la versión del *plugin*, la fecha del reporte, el número de dependencias escaneadas, vulnerabilidades detectadas.

sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates

WebGoat Jenkins
master

Overview
Issues
Security Hotspots
Measures
Code
Activity
More



DEPENDENCY-CHECK

Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in the analysis or otherwise, with regard to the analysis or its use. Any use of the tool and the reporting provided is at the user's risk. In no event shall the copyright holder be liable for any damages arising out of the use of the tool.

[How to read the report](#) | [Suppressing false positives](#) | [Getting Help: github issues](#)

Project: WebGoat

Scan Information ([show all](#)):

- *dependency-check version:* 6.0.1
- *Report Generated On:* Sat, 17 Oct 2020 13:33:13 +0200
- *Dependencies Scanned:* 12093 (9045 unique)
- *Vulnerable Dependencies:* 114
- *Vulnerabilities Found:* 340

Ilustración 82: SonarQube - DependencyCheck Scan Information

Al acceder al detalle de las vulnerabilidades detectadas, se obtiene información como una descripción, el nivel de severidad, referencias, CVE y CWE y la librería o componente afectado.

Published Vulnerabilities

NPM-1486
suppress

Versions of 'http-proxy' prior to 1.18.1 are vulnerable to Denial of Service. An HTTP request with a long body triggers an 'ERR_HTTP_HEADERS_SENT' server. This is only possible when the proxy server sets headers in the proxy request using the 'proxyReq.setHeader' function.

For a proxy server running on 'http://localhost:3000', the following curl request triggers the unhandled exception:

```
curl -XPOST http://localhost:3000 -d "$(python -c 'print("x"*1025)')"
```

Unscored:

- Severity: high

References:

- Advisory 1486: Denial of Service -- [Patch PR](https://github.com/http-party/node-http-proxy/pull/1447/files)

Vulnerable Software & Versions (NPM):

- cpe:2.3:a:*:http-proxy:*:*:1.18.1:*:*:*:*:*

CWE-400: Uncontrolled Resource Consumption ('Resource Exhaustion').(OSSINDEX)
suppress

Ilustración 83: SonarQube - DependencyCheck Published Vulnerabilities

7.4 Jenkins y Docker

La siguiente fase de construcción es la de la creación de la imagen Docker y

posteriormente el contenedor donde se va a desplegar el aplicativo.

```
[Docker] INFO: Step 1/10 : FROM openjdk:11.0.1-jre-slim-stretch
[Docker] INFO:

[Docker] INFO: ---> 49b31a72a85a

[Docker] INFO: Step 2/10 : ARG webgoat_version=v8.1.0
[Docker] INFO:

[Docker] INFO: ---> Running in 5b4bf2058826

[Docker] INFO: ---> 32914342839c

[Docker] INFO: Step 3/10 : RUN apt-get update && apt-get install && useradd --home-dir /home/webgoat --create-home -U webgoat
[Docker] INFO:

[Docker] INFO: ---> Running in 5725fdb26cd1

[Docker] INFO: ---> 2f6bd45328a7

[Docker] INFO: Successfully built 2f6bd45328a7

[Docker] INFO: Successfully tagged webgoat:latest

[Docker] INFO: Build image id:2f6bd45328a7
[Docker] INFO: set portBindings: 9999:8080
[Docker] INFO: created container id 2308fb05b136a435365975d5606c8b628360c8e7ca4dda1906891422f17806b (from image webgoat)
[Docker] INFO: started container id webgoat_container
```

Ilustración 84: Jenkins – Docker Build Image

Se ha creado una nueva imagen con el nombre que se ha indicado en Jenkins:

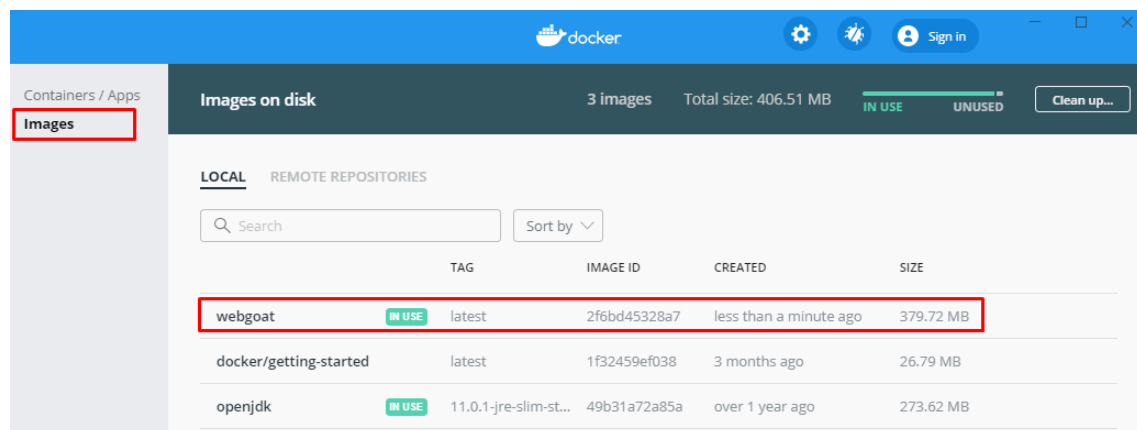


Ilustración 85: Docker Webgoat Image

Y en el apartado *Containers/Apps* también se ha añadido el contenedor “webgoat_cotainer” y lo ha levantado en el puerto 9999 como se puede observar en el recuadro rojo de la siguiente imagen:



Ilustración 86: Docker Webgoat Container

Para comprobar que todo ha ido bien, se accede a la dirección web dónde está corriendo la aplicación WebGoat: <http://localhost:9999/WebGoat>

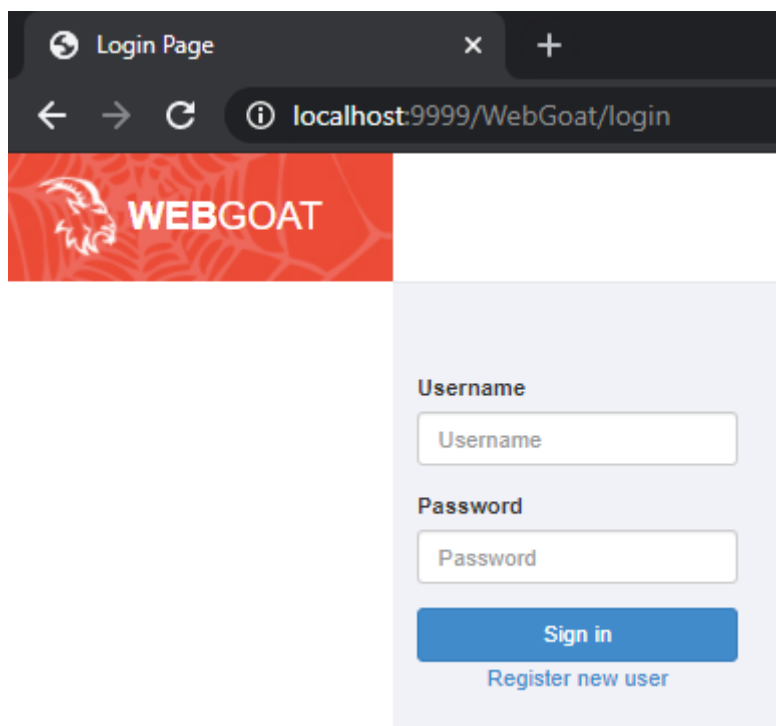


Ilustración 87: WebGoat Login Page

Al registrar un usuario con una contraseña ya se puede navegar por el aplicativo para conocer las diez vulnerabilidades más importantes del Proyecto OWASP 2017 y realizar los ejercicios propuestos.

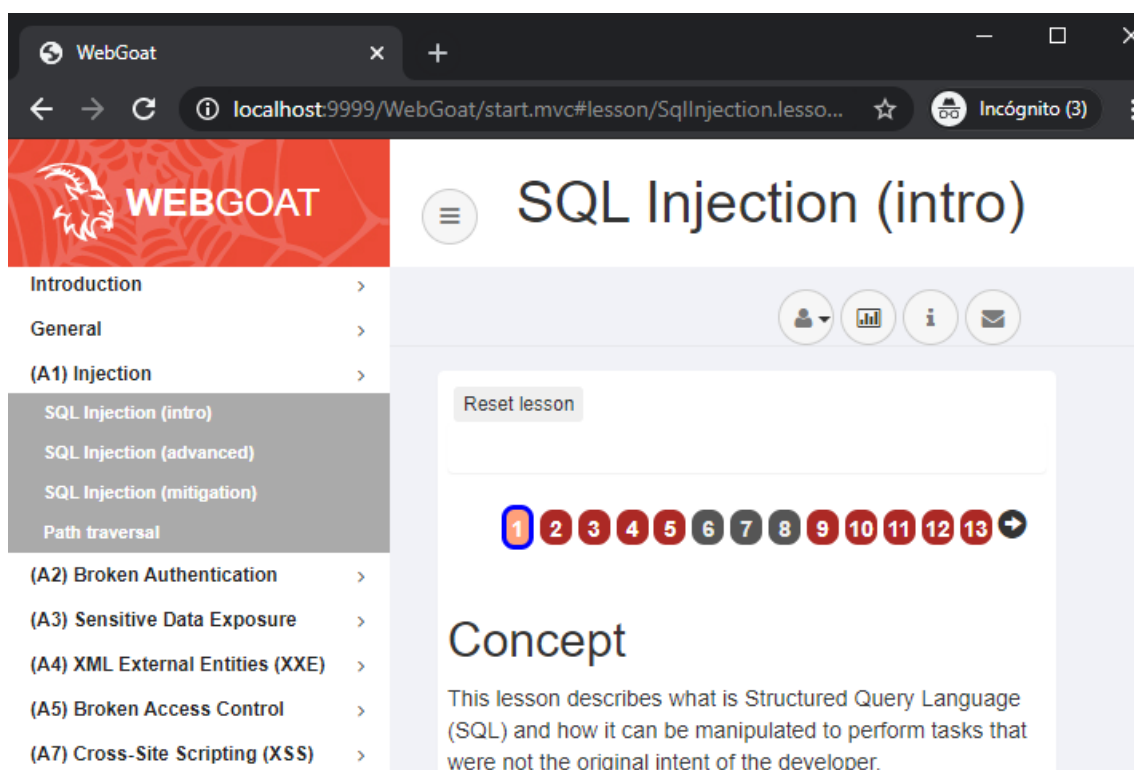


Ilustración 88: WebGoat Navigation

7.5 Jenkins y MailDev

Una vez finalizado la *build* del proyecto, se envía un email a la dirección de correo configurada en el que se indica si la compilación ha sido satisfactoria o ha fallado.

```
Email was triggered for: Always
Email was triggered for: Success
Sending email for trigger: Always
Sending email to: devops@jenkins.com
Sending email for trigger: Success
An attempt to send an e-mail to empty list of recipients, ignored.
Finished: SUCCESS
```

Ilustración 89: Jenkins – Output MailDev

Y en la interfaz web de MailDev aparece el email con el mensaje de que la compilación ha sido exitosa. También se proporciona el número de la *build* y un botón para visualizar la consola de logs del proceso de compilación.

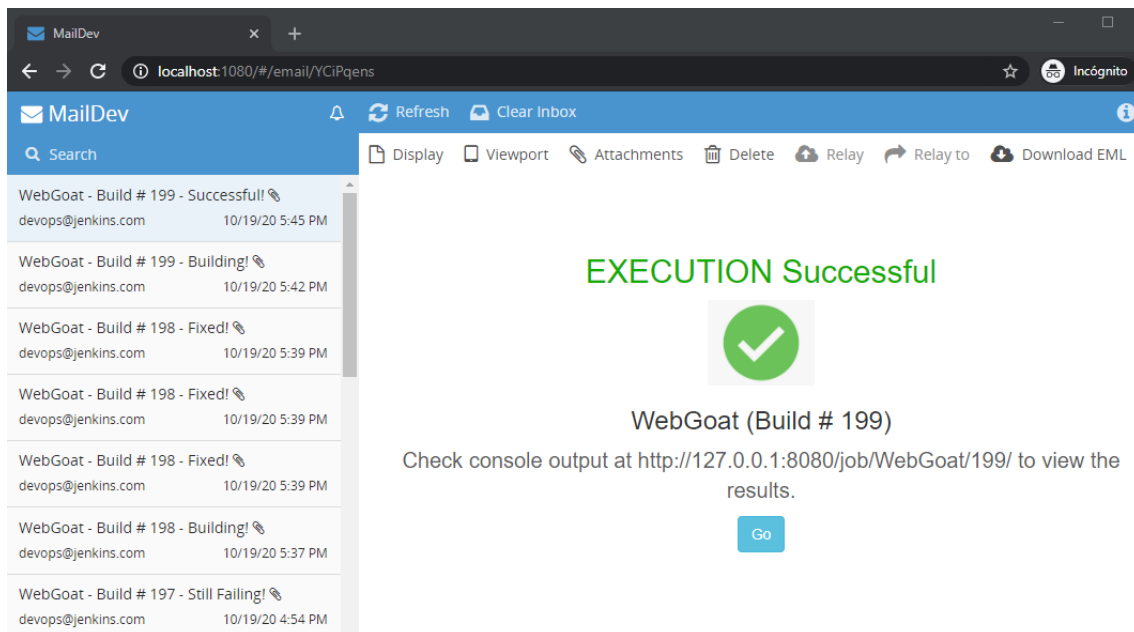


Ilustración 90: MailDev Execution Successful

7.6 Jenkins y OWASP ZAP

La última fase de construcción del proyecto es el análisis dinámico (DAST) de la aplicación en busca de vulnerabilidades en ejecución del aplicativo.

Este análisis es importante para identificar riesgos y amenazas, y vulnerabilidades a nivel de aplicación ya que simula ataques al sitio web que podrían ser realizados por atacantes reales.

```
[ZAP Jenkins Plugin] START BUILD STEP

[ZAP Jenkins Plugin] PLUGIN VALIDATION (PLG), VARIABLE VALIDATION AND ENVIRONMENT INJECTOR EXPANSION (EXP)

  ZAP INSTALLATION DIRECTORY = [ C:\Program Files\OWASP\Zed Attack Proxy ]
  (EXP) HOST = [ localhost ]
  (EXP) PORT = [ 9999 ]
  (EXP) CONTEXT NAME = [ ZAPWebGoat ]
  (EXP) INCLUDE IN CONTEXT = [ http://localhost:9999/WebGoat/* ]
  (EXP) EXCLUDE FROM CONTEXT = [ ]
  (EXP) STARTING POINT (URL) = [ http://localhost:9999/WebGoat/ ]
  (EXP) REPORT FILENAME = [ JENKINS_ZAP_VULNERABILITY_REPORT ]

[ZAP Jenkins Plugin] CONFIGURE RUN COMMANDS for [ C:\Program Files\OWASP\Zed Attack Proxy\zap.bat ]
[ZAP Jenkins Plugin] EXECUTE LAUNCH COMMAND
[Zed Attack Proxy] $ "C:\Program Files\OWASP\Zed Attack Proxy\zap.bat" -daemon -host localhost -port 9999
-config api.key=ZAPROXY-PLUGIN -dir "C:\Users\José María Acuña\.jenkins\workspace\ZAP_CLI"

[ZAP Jenkins Plugin] INITIALIZATION [ START ]

[ZAP Jenkins Plugin] INITIALIZATION [ SUCCESSFUL ]
```

Ilustración 91: Jenkins - Output Zap

Una vez finalizado el análisis dinámico de la aplicación, genera el reporte en un fichero en formato *html* en la ruta “*reports*” de la carpeta *workspace* de Jenkins “**JENKINS_ZAP_VULNERABILITY_REPORT.html**”

La herramienta ZAP ha detectado una vulnerabilidad de tipo High (*SQL Injection*), una de tipo Medium (*Parameter Tampering*), tres de tipo Low (*Absence of Anti-CSRF Tokens*, *Cookie Without SameSite Attribute* y *Cookie No HttpOnly Flag*). Además, ha detectado dos evidencias de carácter informativo (*Timestamp Disclosure – Unix* y *Loosely Scoped Cookie*).

El informe presenta una descripción del tipo de vulnerabilidad, la url afectada, posibles remediaciones, y el CWE (*Common Weakness Enumeration*) correspondiente, si aplica.

Es tarea del auditor de seguridad realizar un *pentest* manual para tratar de verificar algunas de estas vulnerabilidades.

ZAP Scanning Report

Summary of Alerts

Risk Level	Number of Alerts
High	1
Medium	1
Low	3
Informational	2

Alert Detail

High (Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://localhost:9999/WebGoat/register.mvc
Method	POST
Parameter	agree
Attack	agree' OR '1'='1' --
Instances	1

Ilustración 92: ZAP Scanning Report

7.7 Jenkins y Jira

Una vez ha finalizado la compilación del *job* WebGoat, se llama al proceso hijo Jira y se lanza la compilación de este proyecto.

Lanzando una nueva ejecución de Jira
Finished: SUCCESS

Al acceder al *output* de la consola se observa que la ejecución de curl ha sido exitosa. En la siguiente imagen aparecen todas las cabeceras de la petición y la salida del nuevo *ticket* insertado cuyo id es 10011.

```
Server: AtlassianProxy/1.15.8.1
cache-control: no-cache, no-store, no-transform
Content-Type: application/json;charset=UTF-8
Strict-Transport-Security: max-age=315360000; includeSubDomains; preload
Date: Sat, 28 Nov 2020 20:55:56 GMT
ATL-TraceId: 12bbf87053055525
x-aquestid: fae7c7a7-af4a-4425-b07d-0dc000b1ca34
x-aaccountid: 5f9e8a5fb38e610071313e6b
X-XSS-Protection: 1; mode=block
Transfer-Encoding: chunked
timing-allow-origin: *
x-envoy-upstream-service-time: 453
X-Content-Type-Options: nosniff
Connection: keep-alive
set-cookie: atlassian.xsrf.token=dfca2cda-b7c6-4a20-8295-2a7531aa2c4f_a00144d345fa7d9da4108102d9dc16aab947b2d7_lin; Path=/; Secure
Expect-CT: report-uri="https://web-security-reports.services.atlassian.com/expect-ct-report/global-proxy", enforce, max-age=86400

100 393 0 101 100 292 101 292 0:00:01 0:00:01 --:--:-- 292
100 393 0 101 100 292 101 292 0:00:01 0:00:01 --:--:-- 288
{"id":"10011","key":"WEB-6","self":"https://tfm-devsecops.atlassian.net/rest/api/latest/issue/10011"}
C:\Users\Jos, Marja Acuña\.jenkins\workspace\Jira>exit 0
Finished: SUCCESS
```

Para comprobar que se ha añadido la nueva *issue*, refrescamos la página de Jira y se observa que aparece la notificación de “*Build Success*” con la clave WEB-6.

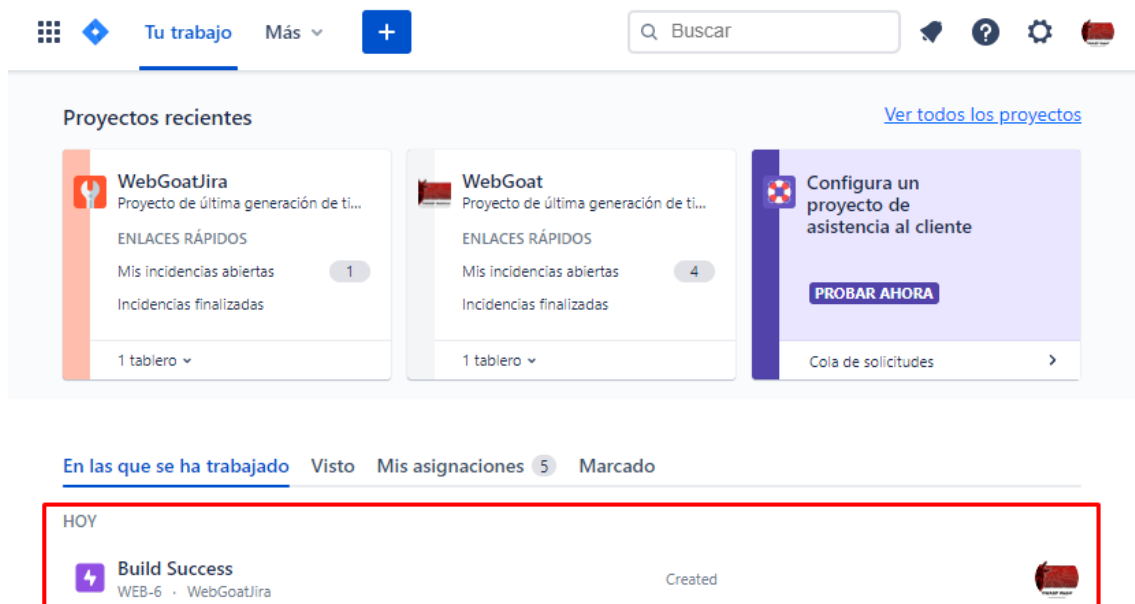


Ilustración 95: Jira Notification

Además, Jira proporciona una *app* para dispositivos móviles. Se ha procedido a la instalación en un *Smartphone* y en la opción Incidencias, aparece la *issue* que acabamos de añadir.

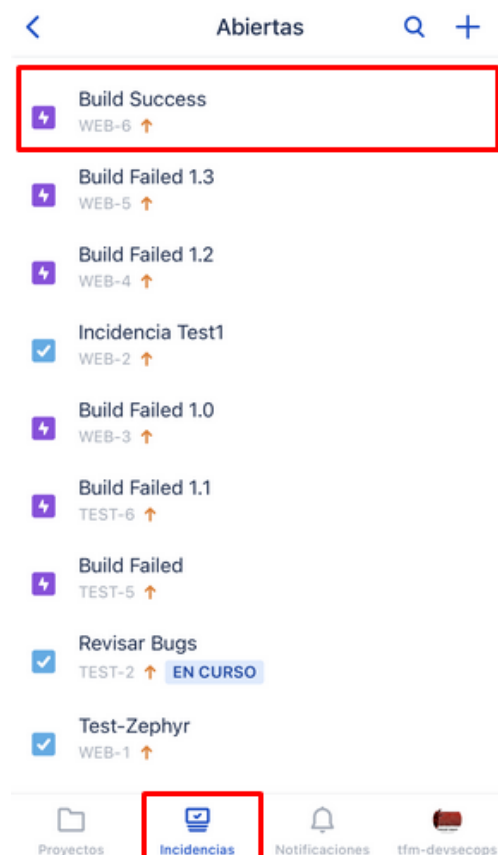


Ilustración 96: Jira App

Al pulsar en la nueva *issue*, la *app* muestra información detallada sobre la misma:

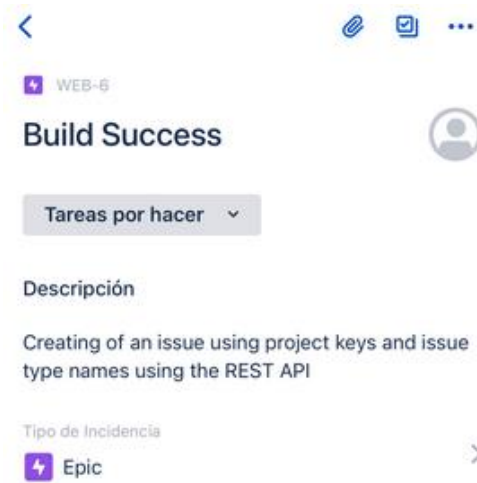


Ilustración 97: Jira Detail App

8. Conclusiones

8.1 Conclusiones generales

El desarrollo de este Trabajo Fin de Máster ha servido para entender cómo encaja el modelo de desarrollo seguro en el ciclo de vida de desarrollo del *software* y por qué es tan necesario en todas sus etapas.

Queda subrayado que la incorporación de medidas de seguridad en las primeras fases del desarrollo de *software* supone un ahorro en costes generales para cualquier organización y que la seguridad debe adoptar un enfoque de responsabilidad compartida para todos los miembros de los equipos IT: seguridad, desarrollo y operaciones.

Se ha demostrado que utilizando herramientas *free* y/o *open source* es posible implementar un ciclo de vida de desarrollo de *software* (SDLC) de forma eficiente, rápida y segura en cualquier entorno empresarial.

También ha quedado patente que es absolutamente necesaria la implantación de procesos de automatización de tareas para conseguir el objetivo de lanzamiento de aplicaciones de *software* de manera rápida y sostenible, ajustándose a los estándares de seguridad de los datos.

Con el propósito de cumplir los objetivos marcados en este trabajo y a efectos de implementar un sistema de integración continua, hemos utilizado Jenkins como herramienta para la ejecución de tareas automáticas como la descarga de las fuentes desde el control de versiones (*checkout* del código subido a GitHub), su posterior compilación, despliegue del proyecto con Docker, análisis de calidad y seguridad de código con SonarQube y dependencias del proyecto con Dependency-Checks, análisis dinámico con OWASP ZAP y *feedback* haciendo uso del servidor de correo Maildev y notificación a través del software Jira.

Se trata de un proceso iterativo que se repetirá tantas veces como sea necesario hasta que la aplicación pueda subirse a un entorno de producción sin errores de seguridad y de forma estable.

8.2 Conclusiones personales

A nivel personal, la labor de investigación y la puesta en funcionamiento del laboratorio han sido esenciales para adquirir una visión global del tipo de herramientas de software libre existentes en el mercado y su papel en la implantación de la metodología DevSecOps.

Se ha utilizado SonarQube como solución de código abierto para ayudar a identificar errores y prevenir vulnerabilidades al principio del proceso de desarrollo, antes de la ejecución de código.

Una vez analizados los resultados del escáner de SonarQube en la aplicación WebGoat desarrollada en java, se puede concluir:

- la herramienta ha detectado un gran número de vulnerabilidades que hay que revisar e interpretar manualmente para eliminar falsos positivos o confirmar si la vulnerabilidad es explotable y cuál es su gravedad en el contexto de la aplicación.

En combinación con el análisis estático, se han realizado pruebas dinámicas con la herramienta OWASP ZAP. Estas pruebas han sido automatizadas a través de Jenkins. Si bien es cierto que las pruebas estáticas y dinámicas son necesarias para la seguridad de cualquier aplicación, no son suficientes:

- no podemos garantizar que una aplicación web sea 100% segura, pero si queremos aproximarnos a esta cifra, además de los análisis SAST y DAST, se debe llevar a cabo un *pentesting* o test de penetración una vez se ha desplegado la aplicación.
- al tratarse de una aplicación que ha sido desarrollada por nuestro propio equipo, los miembros TI de la empresa deberán realizar este tipo de test, conocido como *pentesting* de caja blanca.

Es decir, desde el punto de vista de la seguridad, se deben aplicar estas tres técnicas: SAST y SCA, DAST y *pentesting* o pruebas de penetración.

Como punto final, habría que preguntarse cuáles son los desafíos futuros a los que se enfrenta DevSecOps en un entorno tecnológico tan cambiante. Por poner algunos ejemplos actuales:

- Transición hacia un modelo de computación en la nube. Los contenedores son un ejemplo de técnica de computación en la nube ampliamente utilizada en la actualidad. Herramientas como Kubernetes serán muy usadas por muchos equipos DevSecOps en un futuro próximo.
- Big Data e Internet de las Cosas (IoT) son artífices del enorme crecimiento en la información y procesamiento de los datos que emplean las organizaciones. La implementación de DevSecOps en el diseño y desarrollo de las bases de datos ayudará a proteger los valiosos recursos de que disponen.
- Inteligencia Artificial (AI) y Machine Learning (ML) son la piedra angular de muchos de los procesos de aprendizaje automatizados. A medida que se desarrollen más herramientas que incorporen AI y ML, los equipos DevSecOps las adoptarán para garantizar la seguridad de sus aplicaciones y productos.

Bibliografía

9. Referencias bibliográficas

- Ali Stouky, B. J. (2020). Test automation tools: A comparison between Selenium, Jenkins. *International Journal of Engineering & Technology*, 370. Retrieved from Test automation tools: A comparison between Selenium, Jenkins and Codeception: <https://www.sciencepubco.com/index.php/ijet/article/download/18880/8648>
- Apache Maven Project. (2020). Retrieved from Apache Maven 3.6.3 is the latest release and recommended version for all users: <http://maven.apache.org/download.cgi>
- Blazquez, D. (2020, Mayo 28). *Hdiv*. Retrieved from All About Static Application Security Testing tools: <https://hdivsecurity.com/bornsecure/what-is-sast-static-application-security-testing/>
- Buchanan, I. (n.d.). *History of DevOps*. Retrieved from History of DevOps: <https://www.atlassian.com/devops/what-is-devops/history-of-devops>
- DevOps. (s.f.). Obtenido de DevOps y el ciclo de vida de las aplicaciones: <https://azure.microsoft.com/es-es/overview/what-is-devops/>
- DevSecOps. (s.f.). Obtenido de DevSecOps y la seguridad de DevOps: <https://www.redhat.com/es/topics/devops/what-is-devsecops>
- Docker. (n.d.). Retrieved from Install Docker Desktop on Windows: <https://docs.docker.com/docker-for-windows/install/>
- Docker Uses Cases. (n.d.). Retrieved from How Docker helps development teams: <https://www.docker.com/use-cases>
- EC-Council. (2020, Octubre 27). Retrieved from Phases of the Secure Software Development Life Cycle (SDLC): <https://blog.eccouncil.org/5-phases-of-the-secure-software-development-life-cycle-sdlc/>
- git. (n.d.). Retrieved from git for Windows: <https://gitforwindows.org/>
- GitHub. (s.f.). Obtenido de TFM-GitHub / WebGoat: <https://github.com/TFM-GitHub/WebGoat>
- GitHub. (n.d.). Retrieved from WebGoat is a deliberately insecure application: <https://github.com/WebGoat/WebGoat>

- Handova, D. (2020, Mayo 13). *The Secure Software Development Life Cycle: Syncing Development and Security*. Retrieved from The Secure Software Development Life Cycle: Syncing Development and Security: <https://devops.com/the-secure-software-development-life-cycle-syncing-development-and-security/>
- Heller, M. (2020, Marzo 9). *What is Jenkins?* Retrieved from The CI server explained: <https://www.infoworld.com/article/3239666/what-is-jenkins-the-ci-server-explained.html>
- Jenkins*. (n.d.). Retrieved from Automated CI/CD with Jenkins: <https://medium.com/avmconsulting-blog/automated-ci-cd-with-jenkins-39b21c7c8035>
- Jenkins*. (n.d.). Retrieved from Thank you for downloading Windows Stable installer: <https://www.jenkins.io/download/thank-you-downloading-windows-installer-stable/>
- Jira REST API examples*. (n.d.). Retrieved from This guide contains different examples of how to use the Jira REST API, including how to query issues, create an issue, edit an issue, and others.: <https://developer.atlassian.com/server/jira/platform/jira-rest-api-examples/>
- Jira Software*. (s.f.). Obtenido de Planifica, supervisa y publica software de primera calidad: <https://www.atlassian.com/es/software/jira/free>
- maildev*. (n.d.). Retrieved from MailDev is a simple way to test your emails during development with an easy to use web interface: <https://hub.docker.com/r/maildev/maildev>
- On Point Technology, LLC*. (n.d.). Retrieved from Reducing the cost of bugs: https://dev.onpointtech.com/reducing_the_cost_of_bugs/
- Open Source SIEM tools*. (n.d.). Retrieved from 10 Best Free and Open-Source SIEM Tools in 2020: <https://www.dnsstuff.com/free-siem-tools>
- OWASP Dependency-Check*. (n.d.). Retrieved from Dependency-Check is a Software Composition Analysis (SCA) tool: <https://owasp.org/www-project-dependency-check/>
- OWASP WebGoat*. (s.f.). Obtenido de Learn the hack - Stop the attack: <https://owasp.org/www-project-webgoat/>

OWASP ZAP. (n.d.). Retrieved from Download ZAP 2.9.0:
<https://www.zaproxy.org/download/>

Research Report. (2019). Retrieved from Vulnerability and threat trends:
https://lp.skyboxsecurity.com/rs/440-MPQ-510/images/Skybox_Report_Vulnerability_and_Threat_Trends_2019.pdf

Security - Api Tokens Jira. (n.d.). Retrieved from A primary use case for API tokens is to allow scripts to access REST APIs for Atlassian cloud products using HTTP basic authentication:
<https://id.atlassian.com/manage-profile/security/api-tokens>

SonarQube. (n.d.). Retrieved from Thank you for downloading SonarQube Community Edition: <https://www.sonarqube.org/success-download-community-edition/>

SonarQube User Guide. (n.d.). Retrieved from Concepts: Architecture, Quality:
<https://docs.sonarqube.org/latest/user-guide/concepts/>

SonarScanner. (n.d.). Retrieved from The SonarScanner is the scanner to use when there is no specific scanner for your build system.:
<https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/>

Top 12 Software Development Methodologies: Benefits and Drawbacks. (n.d.). Retrieved from Top 12 Software Development Methodologies: Benefits and Drawbacks: <https://www.velvetech.com/blog/software-development-methodologies/>

What is Agile. (2020, Noviembre 19). Retrieved from Understanding Agile Methodology and Its Types: <https://www.simplilearn.com/tutorials/agile-scrum-tutorial/what-is-agile>

ZAP Add-ons. (n.d.). Retrieved from The OWASP ZAP desktop user guide:
<https://www.zaproxy.org/docs/desktop/addons/>