
# Estrategia de seguridad para el manejo de vulnerabilidades en el ciclo de integración y despliegue continúo en aplicaciones web desarrolladas bajo la metodología DevOps

**Sebastián Cortés Marulanda**

Instituto Tecnológico Metropolitano

Facultad de ingenierías

Medellín, Colombia

2022

# Estrategia de seguridad para el manejo de vulnerabilidades en el ciclo de integración y despliegue continúo en aplicaciones web desarrolladas bajo la metodología DevOps

**Sebastián Cortés Marulanda**

Tesis o trabajo de investigación presentada(o) como requisito parcial para optar al título de:

**Magister en Seguridad Informática**

Director:

Msc. Gabriel Taborda Blandón

Línea de Investigación:

Ciencias Computacionales

Instituto Tecnológico Metropolitano

Facultad de Ingenierías

Medellín, Colombia

2022

# Resumen

DevOps es una metodología utilizada por las empresas para agilizar los procesos de desarrollo y despliegue de software; esta implica, por parte de la compañía, un esfuerzo para centralizar la comunicación, colaboración e integración entre desarrolladores y administradores de infraestructura. Como esta metodología se enfoca en garantizar que la salida del software a producción se realice en el menor tiempo posible, la implementación de buenas prácticas de seguridad es una actividad que se excluye del proceso, sumado al hecho de que los desarrolladores carecen de formación en el campo de la seguridad. En consecuencia, la verificación del nivel de seguridad del software es una tarea que se realiza cuando la aplicación se encuentra en producción, fase en la cual se puede comprometer su funcionamiento si esta presenta vulnerabilidades.

Por lo anterior, se propone una estrategia de seguridad para identificar, consolidar y manejar vulnerabilidades halladas en el ciclo de integración continua y despliegue continuo dentro la metodología DevOps, brindando una posible solución para cubrir las brechas de seguridad que pueden ser utilizadas para comprometer el correcto funcionamiento de una aplicación. Para lograr los objetivos planteados, es necesario consolidar el resultado del análisis de vulnerabilidades del aplicativo de prueba seleccionado en los puntos establecidos por la estrategia de seguridad desarrollada; así mismo, la implementación de las plataformas de DevOps y SecDevOps para llevar acabo la ejecución de las estrategias de seguridad propuestas, a saber: SAST, análisis de dependencias, análisis de composición de código, análisis de vulnerabilidades en imágenes Docker, análisis en kubernetes y DAST, correspondientes a los procesos de integración continua CI y despliegue continuo CD.

**Palabras clave:** Despliegue continuo, DevOps, integración continua, SecDevOps, seguridad Informática, vulnerabilidades.

# Abstract

DevOps is a methodology used by companies to speed up software development and deployment; this methodology requires an effort by companies themselves to centralize communication, collaboration and integration between developers and infractrusture managers.

This methodology is focused on guaranteeing a software deployment to production in the least amount of time possible. For this reason, good security practices are excluded from the process, also developers lack training in this field. Consequently, the verification of software security level is a task that is carried out when the application is deployed in production, a phase in which its operation can be compromised if it presents vulnerabilities.

Due to the above points, a security strategy is proposed to identify, consolidate and manage vulnerabilities found in the cycle of continuous integration and continuous deployment within the DevOps methodology, providing a possible solution to cover security gaps that can be used to compromise the proper functioning of an application

To achieve the stated objectives, it is necessary to consolidate the result of the vulnerability analysis of the selected test application in the points established by the developed security strategy; likewise, the implementation of the DevOps and SecDevOps platforms to carry out the execution of the proposed security strategies, namely: SAST, dependency analysis, code composition analysis, Docker image vulnerability analysis, Kubernetes analysis and DAST, corresponding to CI continuous integration and CD continuous deployment processes.

**Keywords:** Continuous deployment, DevOps, continuous integration, SecDevOps, cybersecurity, vulnerabilities.

# Contenido

Pág.

**Resumen** ..... **III**

**Lista de figuras** ..... **VIII**

**Lista de tablas** ..... **XVII**

**Lista de abreviaturas** ..... **20**

**Introducción** ..... **23**

**1. Marco Teórico y Estado del Arte** ..... **26**

1.1. Marco teórico ..... 26

1.1.1. Aplicaciones ..... 26

1.1.2. Metodologías para el desarrollo de software ..... 27

1.1.3. DevOps ..... 28

1.1.4. Seguridad en aplicaciones web ..... 34

1.1.5. Seguridad en DevOps - SecDevOps ..... 42

1.2. Estado del arte ..... 50

**2. Metodología** ..... **59**

2.1. Fase 1. Contextualización y caracterización de Sec-DevOps ..... 59

2.1.1. Selección del lenguaje de programación para aplicaciones web ..... 59

2.1.2. Identificación de las vulnerabilidades más comunes en aplicaciones web ..... 60

2.1.3. Herramientas DevOps y SecDevOps con la caracterización del uso de cada una 60

2.1.4. Selección de herramientas para la plataforma DevOps y SecDevOps ..... 61

2.2. Fase 2. Propuesta de la estrategia de seguridad en DevOps ..... 61

2.2.1. Selección de parámetros para el diseño de la estrategia e intervención en el proceso de integración continua y despliegue continuo ..... 62

2.2.2. Diseño de la estrategia de seguridad en DevOps ..... 64

2.3. Fase 3. Implementación de la estrategia de seguridad propuesta en SecDevOps ..... 64

2.3.1. Diseño de la plataforma de DevOps y SecDevOps .....64

2.3.2. Implementación de las plataformas DevOps y SecDevOps .....65

2.4. Fase 4. Validación y verificación de la estrategia de seguridad propuesta en SecDevOps 65

2.4.1. Definición del aplicativo web prueba .....66

2.4.2. Ejecución de pruebas para validar la estrategia de seguridad propuesta .....66

2.4.3. Hallazgos encontrados y resultados obtenidos .....67

**3. Resultados .....67**

3.1. Fase 1. Contextualización y caracterización de SecDevOps .....67

3.1.1. Selección del lenguaje de programación .....67

3.1.2. Caracterización de las herramientas DevOps – SecDevOps .....75

3.1.3. Selección de herramientas para la plataforma DevOps y SecDevOps propuesta 83

3.2. Fase 2. Propuesta de la estrategia de seguridad en DevOps .....90

3.2.1. Selección de parámetros para el diseño de la estrategia de seguridad propuesta e intervención en los procesos de CI y CD .....90

3.2.2. Diseño de la estrategia de seguridad propuesta .....92

3.3. Fase 3. Implementación de la estrategia de seguridad propuesta en SecDevOps .... 106

3.3.1. Diseño de la plataforma de DevOps y SecDevOps .....106

3.3.2. Implementación de las plataformas DevOps y SecDevOps .....109

3.4. Fase 4. Validación y verificación de la estrategia de seguridad propuesta en SecDevOps 111

3.4.1. Definición aplicativo web prueba .....111

3.4.2. Ejecución de pruebas sobre las plataformas implementadas .....113

3.4.3. Hallazgos encontrados y resultados obtenidos .....129

**4. Conclusiones y Recomendaciones ..... 145**

4.1. Conclusiones ..... 145

4.2. Recomendaciones ..... 146

**A. Anexo: Revisión estado de la literatura ..... 148**

**B. Anexo: Metodologías de desarrollo de software clásicas, ágiles, y seguras ..... 150**

**C. Anexo: Instalación de las herramientas para la plataforma DevOps propuesta ..... 162**

**D. Anexo: Instalación de las herramientas para la plataforma SecDevOps propuesta ..... 187**

**E. Anexo: Instalación de las herramientas usadas en la prueba dos (2) ..... 192**

**Bibliografía ..... 197**

# Lista de figuras

**---------- Pág**

Figura 4-1. Ciclo iterativo DevOps .....23

Figura 1-1. Arquitectura de una aplicación web .....27

Figura 1-2. Procesos de DevOps en el desarrollo del software. Adaptada de .....29

Figura 1-3. Ciclo iterativo DevOps. Adaptada de .....31

Figura 1-4. Descripción plataforma DevOps .....32

Figura 1-5. Componentes Integración Continua - Herramientas .....32

Figura 1-6. Componentes Despliegue Continuo - Herramientas .....33

Figura 1-7. Herramientas DevOps para la integración continua CI .....33

Figura 1-8. Herramientas DevOps para el despliegue continuo CD .....33

Figura 1-9. Comparación OWASP Top 10: 2017-2021 .....36

Figura 1-10. Estrategias de seguridad durante el ciclo iterativo de DevOps .....43

Figura 1-11. Funcionamiento servicio DSS .....47

Figura 1-12. Estructura pipelines CDe – A .....53

Figura 1-13. Estructura pipelines CDe – B .....53

Figura 1-14. Arquitectura y módulos del framework de seguridad para DevOps .....55

Figura 1-15. CDP seguro y no seguro con las tácticas de seguridad propuestas .....56

Figura 1-16. Aplicación de seguridad en puntos estratégicos del ciclo de vida del desarrollo de software .....57

Figura 2-1. Fases para el desarrollo del proyecto de grado .....59

Figura 2-2. Estrategias de seguridad en el proceso CI - SecDevOps .....62

Figura 2-3. Estrategias de seguridad en el proceso CD – SecDevOps .....63

Figura 3-1. Resultados de la votación de las tecnologías más importantes para los años 2013/2014/2015 ..... 68
Figura 3-2. Tecnologías más populares en la encuesta realizada en el 2016 ..... 69
Figura 3-3. Tecnologías más populares en la encuesta realizada en el 2017 ..... 69
Figura 3-4. Comportamiento de los lenguajes de programación entre los años 2013-2017 ..... 70
Figura 3-5. Tecnologías más populares en la encuesta realizada en el 2018 ..... 70
Figura 3-6. Tecnologías más populares en la encuesta realizada en el 2019 ..... 71
Figura 3-7. Tecnologías más populares en la encuesta realizada en el 2020 ..... 71
Figura 3-8. Top 10: 2019. Lenguajes de programación de las cuatro fuentes ..... 72
Figura 3-9. Histórico de lenguajes de programación de la comunidad TIOBE ..... 73
Figura 3-10. Top 10 índice TIOBE a junio de 2020 ..... 73
Figura 3-11. Resultado Fase 1 – Actividad 1 ..... 74
Figura 3-12. Parámetros obligatorios para CI y CD de la estrategia de seguridad ..... 90
Figura 3-13. Parámetros transversales para CI y CD de la estrategia de seguridad ..... 91
Figura 3-14. Puntos propuestos para evaluar el parámetro RB – CI ..... 97
Figura 3-15. Pipeline base para CI ..... 99
Figura 3-16. Puntos propuestos para evaluar el parámetro RB – CD ..... 99
Figura 3-17. Pipeline base para CD ..... 101
Figura 3-18. Flujo parámetros de la estrategia de seguridad propuesta ..... 106
Figura 3-19. Componentes de la plataforma DevOps con sus respectivas herramientas ..... 107
Figura 3-20. Diseño plataforma DevOps ..... 107
Figura 3-21. Componentes de la plataforma SecDevOps - CI con sus respectivas herramientas ..... 108
Figura 3-22. Componentes de la plataforma SecDevOps - CD con sus respectivas herramientas 108
Figura 3-23. Diseño plataforma SecDevOps ..... 109
Figura 3-24. Romper el Build “RB” en CI ..... 110

Figura 3-25. Romper el Build “RB” en CD ..... 111

Figura 3-26. Diagrama casos de uso ..... 113

Figura 3-27. Recursos del repositorio usado para la prueba 1 ..... 114

Figura 3-28. Detalle Dockerfile ..... 114

Figura 3-29. Jenkinsfile prueba 1 ..... 116

Figura 3-30. Manifiesto. Definición objeto *deployment* usado en la prueba 1 ..... 116

Figura 3-31. Manifiesto. Definición objeto *service* usado en la prueba 1 ..... 116

Figura 3-32. Archivo usado por SonarQube para el análisis realizado en la prueba 1 ..... 117

Figura 3-33. Manifiesto de kubernetes para el despliegue de la base de datos MySQL ..... 117

Figura 3-34. Creación de la base de datos MySQL en kubernetes ..... 117

Figura 3-35. Ingreso al pod de la base de datos para su configuración ..... 118

Figura 3-36. Creación base de datos “adn_restaurante” ..... 118

Figura 3-37. Visualización objeto service de la base de datos desplegada en kubernetes ..... 118

Figura 3-38. Configuración cadena conexión a la base de datos ..... 119

Figura 3-39. Activación ejecución pipeline en Jenkins ..... 119

Figura 3-40. Ejecución exitosa del pipeline – Prueba 1 ..... 120

Figura 3-41. Ejecutable en el repositorio local de Jenkins ..... 120

Figura 3-42. Visualización estado del pod ..... 120

Figura 3-43. Inicialización aplicativo web ..... 121

Figura 3-44. IP pública para consumir el aplicativo web ..... 121

Figura 3-45. Despliegue exitoso aplicativo web ..... 121

Figura 3-46. Ejecución exitosa pipeline para el análisis SAST ..... 123

Figura 3-47. Resultado SAST – Detalle mínimo ..... 123

Figura 3-48. Detalle escaneo WAS con Qualys – Pentesting ..... 124

Figura 3-49. Ejecución exitosa del escaneo con Nikto – Pentesting.....124
Figura 3-50. Tipos de análisis ofrecidos por Nessus Essentials .....125
Figura 3-51. Resultado ejecución pipeline SecDevOps.....128
Figura 3-52. Detalle resultado pipeline SecDevOps - 1 .....128
Figura 3-53. Detalle resultado pipeline SecDevOps – 2.....128
Figura 3-54. Detalle resultado pipeline SecDevOps – 2.....128
Figura 3-55. Repositorio generado luego de la ejecución del pipeline SecDevOps .....129
Figura 3-56. Reporte generado por DAST.....129
Figura 3-57. Quality Gate “QG” configurado – Clasificación mantenibilidad.....129
Figura 3-58. Resultado de la evaluación del Quality Gate – Cobertura/Mantenibilidad .....130
Figura 3-59. Estado del QG evaluado .....130
Figura 3-60. Puntos análisis Prueba 2 .....131
Figura 3-61. OWASP Top 10: 2021. Comparación con el Top10:2017 .....131
Figura 3-62. Detalle hallazgos encontrados en el análisis SAST .....132
Figura 3-63. Hallazgos severidad alta .....132
**Figura 3-64**. Detalles vulnerabilidad Log4Shell .....133
Figura 3-65. Hallazgos severidad media .....133
Figura 3-66. Hallazgos severidad baja .....133
Figura 3-67. Resultado análisis a: http://20.97.136.205/restaurante/ventas/ .....134
Figura 3-68. Resultado análisis a: http://20.97.136.205/.....134
Figura 3-69. Resultado análisis a: http://20.97.136.205/restaurante/ .....134
Figura 3-70. Resultado análisis de dependencias y composición de código .....137
Figura 3-71. Resultado análisis de dependencias y composición de código - Detalle.....138
Figura 3-72. Resultado análisis estático de código SAST .....138
Figura 3-73. Resultado análisis estático de código SAST –Detalle Security Hotspots .....138

Figura 3-74. Detalle para el hallazgo encontrado ..... 139
Figura 3-75. Resumen evaluación política Anchore ..... 140
Figura 3-76. Reporte evaluación política Anchore ..... 140
Figura 3-77. Reporte CVE dado por Anchore ..... 141
Figura 3-78. Hallazgos críticos y altos ..... 141
Figura 3-79. Hallazgos medios ..... 141
Figura 3-80. Hallazgos bajos ..... 142
Figura 3-81. Reporte CVE dado por Anchore ..... 142
Figura 3-82. Reporte DAST ..... 142
Figura 3-83. Reporte DAST – Detalle ..... 143
Figura 4-1. Flujo de la metodología SCRUM ..... 154
Figura 4-2. Creación redes y subredes en Azure ..... 162
Figura 4-3. Características máquina virtual para la ejecución de Jenkins creada en Azure ..... 162
Figura 4-4. Creación regla en el grupo de seguridad creado ..... 163
Figura 4-5. Comandos para instalar Jenkins en Ubuntu 20.04 ..... 163
Figura 4-6. Comprobación estado Jenkins ..... 164
Figura 4-7. Página inicial de Jenkins para su configuración ..... 164
Figura 4-8. Clave temporal asignada para la configuración. ..... 164
Figura 4-9. Instalación plugins sugeridos por Jenkins ..... 164
Figura 4-10. Creación del usuario administrador de Jenkins ..... 165
Figura 4-11. Pantalla principal Jenkins ..... 165
Figura 4-12. Características máquina virtual para la ejecución de GitLab ..... 166
Figura 4-13. Descarga paquetes GitLab ..... 166
Figura 4-14. Ejecución comando para la instalación de GitLab ..... 167

Figura 4-15. Finalización de la instalación de GitLab .....167
Figura 4-16. Ruta para obtener la clave temporal del usuario root .....168
Figura 4-17. Ejecución comando para iniciar GitLab .....168
Figura 4-18. Acceso a GitLab .....169
Figura 4-19. Pantalla de administración de GitLab .....169
Figura 4-20. Componentes instalados con sus respectivas versiones .....169
Figura 4-21. Versión JDK del Java instalado .....170
Figura 4-22. Configuración de las variables para el despliegue de SonarQube .....170
Figura 4-23. Modificación de los límites del kernel de la vm .....170
Figura 4-24. Creación del usuario sonar con su respectiva contraseña .....170
Figura 4-25. Descarga e instalación de los paquetes de PostreSQL .....171
Figura 4-26. Instalación y verificación de PostgreSQL .....171
Figura 4-27. Creación del usuario postgres con su respectiva contraseña. .....171
Figura 4-28. Validación del usuario postgres .....171
Figura 4-29. Creación del usuario “sonaruser” .....172
Figura 4-30. Creación base de datos “sonardb” .....172
Figura 4-31. Descarga y descomprimir los binarios .....172
Figura 4-32. Movimiento de los binarios y acceso garantizado al usuario sonar. .....173
Figura 4-33. Configuración cadena de conexión a la base de datos “sonardb” .....173
Figura 4-34. Configuración en el ejecutable sonar.sh .....173
Figura 4-35. Creación archivo “sonar.service” .....174
Figura 4-36. Inicio servicio sonar y verificación del estado .....174
Figura 4-37. Consumo de la herramienta por la WEB .....174
Figura 4-38. Configuración instalación Gradle .....175
Figura 4-39. Configuración instalación Gradle .....175

Figura 4-40. Configuración instalación Gradle ..... 175
Figura 4-41. Descarga paquetes Docker ..... 176
Figura 4-42. Visualización versiones descargadas ..... 176
Figura 4-43. Instalación de Docker ..... 176
Figura 4-44. Instalación plugins Docker en Jenkins ..... 177
Figura 4-45. Parámetros de configuración del ACR ..... 177
Figura 4-46. Resumen del ACR creado ..... 177
Figura 4-47. Llaves de acceso del ACR ..... 178
Figura 4-48. Creación credencial para el ACR ..... 178
Figura 4-49. Propiedades a configurar en el clúster de kubernetes – Parte 1 ..... 178
Figura 4-50. Propiedades a configurar en el clúster de kubernetes – Parte 2 ..... 179
Figura 4-51. Propiedades a configurar en el clúster de kubernetes – Parte 3 ..... 179
Figura 4-52. Resumen propiedades del clúster configurado ..... 180
Figura 4-53. Procedimiento para conectar al clúster de kubernetes creado ..... 180
Figura 4-54. Ruta donde se descarga el archivo “config” ..... 180
Figura 4-55. Validación de la descarga del archivo “config” ..... 180
Figura 4-56. Procedimiento para descargar el archivo “config” localmente ..... 181
Figura 4-57. Configuración cliente kubectl ..... 181
Figura 4-58. Plugins instalados en Jenkins ..... 181
Figura 4-59. Credencial para la conexión al clúster de kubernetes configurada ..... 182
Figura 4-60. Generación token en SonarQube ..... 182
Figura 4-61. Generación token en SonarQube ..... 182
Figura 4-62. Configuración SonarQube Scanner – Parte 1 ..... 183
Figura 4-63. Configuración SonarQube Scanner – Parte 2 ..... 183

Figura 4-64. Configuración credencial SonarQube - Jenkins ..... 184
Figura 4-65. Configuración servidor SonarQube ..... 184
Figura 4-66. Propiedades token generado ..... 185
Figura 4-67. Plugins instalados en GitLab ..... 185
Figura 4-68. Credencial creada para GitLab ..... 185
Figura 4-69. Creación a GitLab configurada ..... 186
Figura 4-70. Creación credencial para el usuario root de GitLab ..... 186
Figura 4-71. Plugin instalado en Jenkins. ..... 187
Figura 4-72. Configuración instalación automática de la herramienta. ..... 187
Figura 4-73. Validación instalación docker-compose ..... 188
Figura 4-74. Descarga manifiesto de Anchore en el directorio ..... 188
Figura 4-75. Instalación de Anchore ..... 188
Figura 4-76. Validación contenedores ..... 188
Figura 4-77. Instalación estado Anchore-engine ..... 189
Figura 4-78. Validación instalación anchore-cli ..... 189
Figura 4-79. Validación de Anchore mediante anchore-cli ..... 189
Figura 4-80. Cambio contraseña para el usuario “admin” ..... 189
Figura 4-81. Validación estado de los servicios con el usuario “admin” ..... 189
Figura 4-82. Adición ACR a Anchore ..... 190
Figura 4-83. Instalación plugin de Anchore en Jenkins ..... 190
Figura 4-84. Configuración parámetros para el consumo de Anchore ..... 190
Figura 4-85. Instalación plugin SSH en Jenkins ..... 191
Figura 4-86. Configuración pipeline para ejecución DAST ..... 191
Figura 4-87. Instalación Kubescape ..... 191
Figura 4-88. Inicio sesión AppScan on Cloud ..... 192

Figura 4-89. Pantalla principal AppScan on Cloud ..... 193
Figura 4-90. Creación API desde AppScan On Cloud ..... 193
Figura 4-91. Instalación plugin AppScan on Cloud ..... 193
Figura 4-92. Credencial para AppScan on Cloud ..... 193
Figura 4-93. Registro para obtener el código de activación ..... 194
Figura 4-94. Código de activación ..... 194
Figura 4-95. Descarga e inicio de instalación ..... 195
Figura 4-96. Instalación Nessus ..... 195
Figura 4-97. Pantalla principal Nessus ..... 195
Figura 4-98. Versión Kali Linux usada ..... 196
Figura 4-99. Nikto en Kali Linux ..... 196

# Lista de tablas

**----------Pág**

Tabla 1-1. Integradores CI/CD.....34

Tabla 1-2. Herramientas para implementar seguridad en el ciclo de integración continua].....49

Tabla 1-3. Herramientas para implementar seguridad en el ciclo de despliegue continuo .....49

Tabla 1-4. Componentes claves y riesgos potenciales en un CDP.....56

Tabla 3-1. Clasificación de los ocho lenguajes de programación más constantes .....74

Tabla 3-2. Consolidación de lenguajes de programación orientados a aplicaciones web .....74

Tabla 3-3. Herramientas DevOps – SecDevOps con su caracterización .....75

Tabla 3-4. Criterios de selección para el integrador CI/CD.....83

Tabla 3-5. Criterios de selección para el repositorio centralizado de código .....84

Tabla 3-6. Criterios de selección para el gestor de paquetes-compilación.....84

Tabla 3-7. Criterios de selección para el repositorio de artefactos.....84

Tabla 3-8. Criterios de selección para el componente de verificación de calidad de código.....84

Tabla 3-9. Criterios de selección para el componente de pruebas unitarias .....85

Tabla 3-10. Criterios de selección para el componente de configuración y virtualización de entornos .....85

Tabla 3-11. Herramientas seleccionadas para la plataforma DevOps.....85

Tabla 3-12. Criterios de selección para el componente de SAST - CI .....86

Tabla 3-13. Criterios de selección para el componente de análisis de dependencias - CI.....86

Tabla 3-14. Criterios de selección para el componente de análisis de vulnerabilidades en imágenes Docker - CD .....87

Tabla 3-15. Criterios de selección para el componente de DAST - CD .....87

Tabla 3-16. Herramientas seleccionadas para una plataforma SecDevOps estándar - CI .....87

Tabla 3-17. Herramientas seleccionadas para una plataforma SecDevOps estándar - CD.....87

Tabla 3-18. Criterios de selección para la herramienta del componente: precommit (Administración de secretos).....88

Tabla 3-19. Criterios de selección para la herramienta del componente: Análisis de composición de software .....88

Tabla 3-20. Criterios de selección para la herramienta del componente: Análisis manifiestos Kubernetes.....89

Tabla 3-21. Herramientas seleccionadas para los controles de seguridad seleccionados en el proceso CI .....89

Tabla 3-22. Herramienta seleccionada para los controles de seguridad seleccionados en el proceso CD.....89

Tabla 3-23. Clasificación vulnerabilidades.....96

Tabla 3-24. Propuesta de la cantidad de vulnerabilidades permitidas por el parámetro RB .....96

Tabla 3-25. Siglas Matriz RACI .....102

Tabla 3-26. Matriz RACI propuesta equipo para una empresa pequeña o pyme .....103

Tabla 3-27. Matriz RACI propuesta equipo para una grande empresa .....104

Tabla 3-28.Propuesta de la cantidad de vulnerabilidades permitidas por el parámetro RB para una empresa grande .....104

Tabla 3-29. Endpoints expuestos por el aplicativo web .....123

Tabla 3-30. Resultados análisis con la herramienta Nikto.....134

Tabla 3-31. Resultados análisis con la herramienta Qualys .....135

Tabla 3-32. Resultados análisis con la herramienta Nessus .....136

Tabla 3-33. Hallazgos en el CI de: Análisis dependencias y composición de código –Prueba 3 ....139

Tabla 3-34. Hallazgos en el CI de: Análisis estático de código –Prueba 3 .....140

Tabla 3-35. Hallazgos en el CD de: Análisis seguridad de imágenes Docker –Prueba 3.....143

Tabla 3-36. Hallazgos en el CD de: Análisis seguridad en kubernetes –Prueba 3 .....144

Tabla 3-37. Hallazgos en el CD de: Análisis dinámico de código DAST –Prueba 3 .....144

Tabla 4-1. Artículos iniciales, artículos luego de la valoración de la calidad .....148

Tabla 4-2. Detalle artículos valoración de la calidad .....148

# Lista de abreviaturas

## Abreviaturas

| Abreviatura | Término |
|---|---|
| *ACR* | Azure Container Registry |
| *AKS* | Azure Kubernetes Service |
| *API* | Application Programming Interfaces |
| *ASVS* | Application Security Verification Standard |
| *AWS* | Amazon Web Services |
| *CD* | Continuous Deployment |
| *CDe* | Continuous Delivery |
| *CI* | Continuous Integration |
| *CPU* | Central Processing Unit |
| *CVE* | Common Vulnerabilities and Exposures |
| *CVSS* | Common Vulnerability Scoring System |
| *CWE* | Common Weakness Enumeration |
| *DAST* | Dynamic Application Security Testing |
| *DB* | DataBase |
| *DEV* | Development |
| *DevOps* | Development & Operations |

| Abreviatura | Término |
|---|---|
| *DNS* | Domain Name System |
| *DSS* | Docker Security Scanner |
| *HTTP* | Hypertext Transfer Protocol |
| *HTTPS* | Hypertext Transfer Protocol Secure |
| *GCP* | Google Cloud Computing |
| *IAST* | Interactive Application Security Testing |
| *IDE* | Integrated Development Environment |
| *JAR* | Java ARchive |
| *JDK* | Java Development Kit |
| *NSG* | Network Security Group |
| *NVD* | National Vulnerability Database |
| *QG* | Quality Gate |
| *OCD* | Origin Continuous Deployment |
| *OPS* | Operations |
| *OS* | Open Source |
| *OSS* | Open Source Software |
| *OWASP* | Open Web Application Security Project |
| *OWASP ZAP* | Zed Attack Proxy |

| Abreviatura | Término |
|---|---|
| *RAM* | Random Access Memory |
| *RASP* | Runtime Application Self Protection |
| *RB* | Romper el Build |
| *SAST* | Static Application Security Testing |
| *SecDevOps* | Security & Development & Operations |
| *SCA* | Software Composition Analysis |
| *SCM* | Source Code Management |
| *SFK* | Security Knowledge Frameworks |
| *SQL* | Structured Query Language |
| *SSH* | Secure Shell |
| *URL* | Uniform Resource Locator |
| *vCPU* | Virtual Central Processing Unit |
| *VM* | Virtual Machine |
| *VNET* | Virtual Network |
| *WAS* | Web Application Vulnerability Scan |
| *WWW* | World Wide Web |

# Introducción

**Objetivos**

**Objetivo General**

"Proponer una estrategia de seguridad que detecte vulnerabilidades en el ciclo de integración y despliegue continúo en el desarrollo de aplicaciones web bajo la metodología DevOps, para reducir posibles brechas de seguridad que puedan comprometer su correcto funcionamiento al ingresar a producción".

**Objetivos Específicos**

- "Seleccionar uno de los lenguajes de programación más populares sobre los cuales se desarrollan las aplicaciones web".
- "Identificar las herramientas open source necesarias para la implementación de una plataforma que permita la identificación y manejo de vulnerabilidades en el CI y CD dentro de la metodología DevOps, así como identificar las posibles vulnerabilidades estándares en un aplicativo web con base en el lenguaje de programación seleccionado".
- "Validar la estrategia de seguridad sobre un aplicativo de prueba, durante los ciclos CI/CD que permita hallar el nivel de seguridad".

DevOps es una metodología empleada por las empresas para la agilización de los procesos de desarrollo y despliegue de software; esta metodología está compuesta por las fases: plan, código, construcción y pruebas, conformando el proceso de integración continua CI, el cual es el encargado de planear y construir el código y las fases del proceso de despliegue o entrega continua: lanzamiento, despliegue, operación y monitoreo responsable de la puesta en producción del software. Para un mayor entendimiento de lo dicho anteriormente, se expone la Figura 4-1.

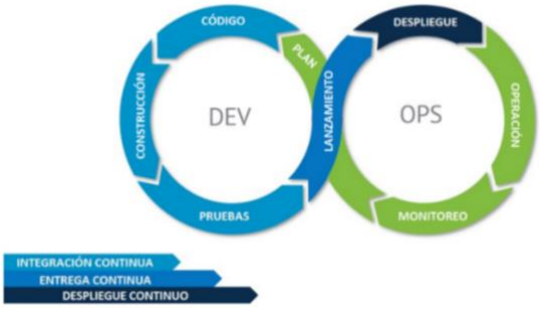



**Figura 4-1.** Ciclo iterativo DevOps

Como esta metodología se enfoca en garantizar que la salida del software a producción se realice en el menor tiempo posible, la implementación de buenas prácticas de seguridad es una actividad que se excluye del proceso, sumado al hecho de que los desarrolladores carecen de formación en el campo de la seguridad. En consecuencia, la verificación del nivel de seguridad del software es una tarea que se realiza cuando la aplicación se encuentra en producción, fase en la cual se puede comprometer su funcionamiento si presenta vulnerabilidades. Teniendo en cuenta lo mencionado anteriormente y basados en la información tomada de la literatura, se menciona la metodología SecDevOps como la encargada de implementar diferentes controles mediante estrategias de seguridad tales como pruebas de seguridad de aplicaciones estáticas “SAST”, análisis de dependencias y pruebas de seguridad de aplicaciones dinámicas “DAST”, en los procesos de integración y despliegue continuos a lo largo del flujo DevOps. A partir del análisis realizado, se evidenció que la metodología carece de puntos tanto en lo técnico, como en lo metodológico y lo cultural.

Por lo anterior, se propone una estrategia de seguridad para identificar, consolidar y manejar vulnerabilidades halladas en el ciclo de integración continua y despliegue continuo dentro la metodología DevOps, brindando una posible solución para cubrir las brechas de seguridad que pueden ser utilizadas para comprometer el correcto funcionamiento de una aplicación. Partiendo de esto, se construyó tanto una estrategia de seguridad integral, flexible y adaptable al tamaño de la empresa y al nivel de madurez del equipo de seguridad que la desee implementar, como la plataforma SecDevOps, empleando herramientas de software libre y servicios de Azure. La estrategia se compone de siete parámetros clasificados en obligatorios y transversales, donde los obligatorios son: implementación de controles de seguridad adicionales, adopción del parámetro romper el build o RB, el cual hace referencia a la acción a tomar para detener el ciclo de vida del software debido a un fallo o violación en los umbrales de seguridad establecidos y construcción de pipelines base para CI y CD; por parte de los parámetros transversales se tiene: gestión y administración de vulnerabilidades halladas, definición roles y responsabilidades, adopción cultura SecDevOps y monitoreo continuo.

**Contenido del documento**

***Marco teórico y estado del arte:*** en este apartado se exponen todos los conceptos fundamentales e indispensables para comprender el desarrollo de esta tesis. Además, se comparten los diferentes trabajos relacionados a la implementación de seguridad en DevOps que apoyaron a la elaboración y construcción de esta tesis.

***Metodología:*** capítulo donde se expone el cómo se trabajó cada una de las actividades de las fases propuestas para dar cumplimiento a los tres objetivos específicos planteados inicialmente.

***Resultados:*** en esta sección se exponen de manera detallada las evidencias, imágenes y fragmentos de código obtenidos luego de la ejecución de las actividades descritas en el apartado de la metodología.

***Conclusiones y Recomendaciones:*** en este apartado se comparten las conclusiones identificadas luego de cumplir cada uno de los objetivos específicos y, así mismo, las sugerencias para futuros trabajos que se puedan realizar a partir del desarrollo de esta tesis.

***Anexos:*** como insumo significativo, en esta sección se comparten los procedimientos necesarios para desplegar las diferentes herramientas y plataformas descritas en la tesis.

***Bibliografía:*** finalmente, se comparten las referencias empleadas en la elaboración del documento final de la tesis propuesta.

# 1. Marco Teórico y Estado del Arte

En este capítulo se describen los conceptos necesarios para abordar y comprender el desarrollo de este trabajo de grado, dichos conceptos se agrupan dentro de las siguientes categorías: aplicaciones, DevOps, seguridad en aplicaciones, seguridad en DevOps (SecDevOps). Además, se exponen y se mencionan trabajos que abordaron objetivos similares a este.

## 1.1. Marco teórico

En este apartado se mencionan los conceptos más relevantes para el desarrollo de esta investigación, los cuales ayudarán a comprender el desarrollo de la misma.

### 1.1.1. Aplicaciones

Se describen, a continuación, algunos aspectos generales e importantes a tener en cuenta sobre las aplicaciones, tipos de aplicaciones, aplicaciones web y su arquitectura.

- ***Aplicaciones y tipos de aplicaciones***

Una aplicación es un programa informático diseñado como una herramienta para realizar operaciones o funciones específicas, enfocada en facilitar tareas complejas y brindar una mejor experiencia informática para los usuarios. Conservan una diferencia respecto de los sistemas operativos o lenguajes de programación, ya que estas están construidas para cumplir una función específica [1]. Entre los tipos de aplicaciones, se tienen las aplicaciones de escritorio, aplicaciones móviles y aplicaciones web.

- ***Aplicaciones de escritorio***

Son aquellas aplicaciones que dependen de un software instalado directamente en la estación de trabajo de cada persona que requiere hacer uso del sistema. Normalmente, la comunicación a la base de datos se realiza por medio de internet [2].

- ***Aplicaciones móviles***

Las aplicaciones móviles son programas diseñados para ser ejecutados en teléfonos, tablets y otros dispositivos móviles, que permiten al usuario realizar ciertas actividades, acceder a servicios, consultar información, entre otras actividades. Android, IOS, y Windows Phone, son de los sistemas operativos más

importantes para este tipo de aplicaciones. Cada uno cuenta con sus respectivas tiendas para descargar e instalar alguna aplicación requerida [3].

- ***Aplicación WEB***

Una aplicación web es una aplicación informática o tipo de software que está basada en un código de lenguaje de programación como HTML, JavaScript, CSS, entre otros; la cual está diseñada para ser soportada desde un navegador y accedida vía web en una red interna o externa. Siguiendo esto, se define como un programa informático basado en un lenguaje que se ejecuta en un entorno del navegador web. La aplicación web es basada en una arquitectura cliente/servidor, en donde el cliente es un navegador web y el servidor es un host o máquina en donde se aloja la aplicación, la cual puede ser presentada mediante los protocolos, HTTP (Hypertext Transfer Protocol), y el protocolo seguro HTTPS (Hypertext Transfer Protocol Secure), lo que lleva a que el trabajo es realizado en los servidores donde está alojada la aplicación web y las bases de datos [4]. En la siguiente Figura 1-1 se observa la arquitectura básica de una aplicación web.

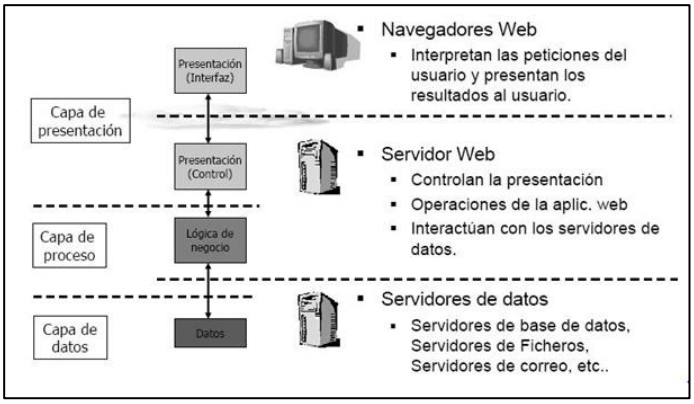



**Figura 1-1.** Arquitectura de una aplicación web **[5]**

## 1.1.2. Metodologías para el desarrollo de software

***Metodologías para el desarrollo de software***

Una metodología de desarrollo de software es un marco de trabajo que se usa para constituir, programar e inspeccionar el proceso de desarrollo de un sistema de información [6]. Las metodologías de desarrollo de software clásicas, ágiles y seguras se presentan en el *Anexo: Metodologías de desarrollo de software clásicas, ágiles, y seguras*.

### 1.1.3. DevOps

Es una metodología que se centraliza en mejorar la comunicación entre los diferentes equipos de operaciones y los desarrolladores, para así abandonar los flujos inoperativos y revisiones continuas del trabajo de unos y otros, para empezar a trabajar de forma unidireccional y colaborativa.

En este sentido, DevOps es la combinación de los términos *Dev (Development)* y *Ops (Operations)*, designa la unión de los roles (desarrollo, operaciones de TI, ingeniería de la calidad y seguridad) para que se coordinen y colaboren para producir productos mejores y más confiables. Al adoptar una cultura de DevOps junto con prácticas y herramientas de DevOps, los equipos adquieren la capacidad de responder mejor a las necesidades de los clientes, aumentar la confianza en las aplicaciones que crean y alcanzar los objetivos empresariales tales como: adaptación al mercado y a la competencia, mantenimiento de la estabilidad y confiabilidad del sistema y mejora del tiempo medio de recuperación logrando esto en un corto tiempo [7].

Aunque la adopción de DevOps automatiza y optimiza los procesos con tecnología, todo comienza con la cultura interna de la organización y las personas que participan en ella. El desafío de crear y mantener la cultura requiere cambios profundos en la forma en las que las personas trabajan y colaboran, hasta el punto que se tenga un entorno que facilite el desarrollo de equipos de alto rendimiento [7].

- ***Procesos de DevOps***

A continuación, se relacionan los tres procesos necesarios para cumplir con todo el ciclo del desarrollo de una aplicación dentro de DevOps, dichas prácticas permiten a los colaboradores de TI liberar de manera más frecuente, software fiable de calidad.

***Integración continua***

CI, Continuous Integration. Es una práctica de desarrollo ampliamente establecida en la industria de desarrollo de software, en la que los miembros de un equipo integran y fusionan código con frecuencia y durante varias veces al día. CI permite a las empresas de software tener ciclos de lanzamiento más cortos y frecuentes, mejorar la calidad del software y aumentar la productividad de sus equipos. Esta práctica incluye la creación y prueba de software automatizado [8]. La integración continua pone un gran énfasis en la automatización de pruebas para verificar que la aplicación no se interrumpa cada vez que se integran nuevas funcionalidades en la rama principal del código [9].

***Entrega Continua***

CDE, Continuous Delivery. La entrega continua es una práctica de desarrollo de software mediante la cual se preparan automáticamente los cambios en el código y se entregan a la fase de producción. La entrega continua implementa todos los cambios en el código realizados en la integración continua en un ambiente de pruebas o producción después de la fase compilación. Además, permite a los desarrolladores automatizar las pruebas más allá, por lo que pueden verificar las actualizaciones en varios entornos antes de enviarlas al cliente, esto siendo una gran ventaja, ya que pueden hacer una validación más completa y hallar problemas por anticipado [10].

***Despliegue/Implementación Continua***

CD, Continuous Deployment. La implementación continua cuenta con una ventaja respecto de la entrega continua. Con esta práctica, todos los cambios que pasan por las etapas de su canal de producción se entregan a sus clientes. No hay intervención humana y solo una prueba fallida evitará que se implemente un nuevo cambio en producción. La implementación continua es una excelente manera de acelerar el ciclo de retroalimentación con sus clientes y aliviar la presión del equipo, ya que no hay un día específico para el lanzamiento. Los desarrolladores pueden concentrarse en la creación de software y ven que su trabajo se activa minutos después de haber terminado de trabajar en él [9].

Los tres (3) procesos implementadas en DevOps se puede observar en la Figura 1-2.

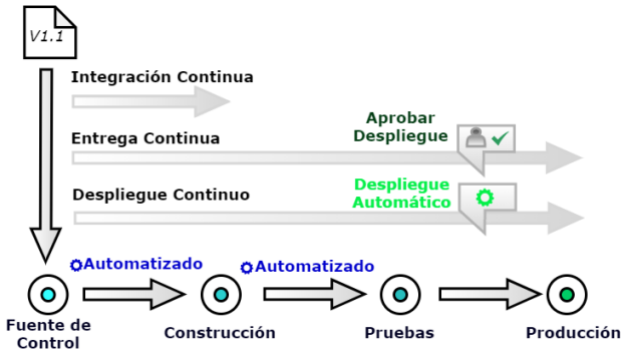



**Figura 1-2.** Procesos de DevOps en el desarrollo del software. Adaptada de **[10]**

El ciclo de vida iterativo en DevOps consta de las siguientes etapas o fases: *Plan – Code - Build – Test – Release – Deploy – Operate – Monitor*.

- **Planeación (Plan):** en la fase de planeamiento, los equipos de DevOps conciben, definen y describen las características y la funcionalidad de las aplicaciones y los sistemas que van a crear.

Realizan un seguimiento del progreso tanto de forma general como de forma detallada, desde tareas de un único producto hasta tareas que abarcan numerosos productos. Se utilizan metodologías como scrum y kanban para crear registros de trabajo pendiente, seguimiento de errores y la visualización del progreso.

- **Código (Code):** en esta fase se realiza el diseño del software, la creación del código, visión, integración del código por parte de los miembros del equipo y la compilación de ese código en artefactos de compilación que se pueden implementar en varios entornos.
- **Construcción (Build):** en esta fase de DevOps, el desarrollo de software se lleva a cabo constantemente, dividiendo todo el proceso de desarrollo en pequeños ciclos. Esto con el fin de beneficiar al equipo de DevOps para acelerar el desarrollo de software y el proceso de entrega.
- **Pruebas (Test):** el equipo de pruebas usa herramientas para identificar y corregir errores en el nuevo código de manera continua.
- **Lanzamiento (Release):** en esta fase, la nueva funcionalidad se integra con el código existente y se llevan a cabo las pruebas. El desarrollo continuo solo es posible gracias a la integración y las pruebas continuas.
- **Despliegue (Deploy):** en esta etapa, el proceso de implementación se realiza continuamente. Se ejecuta de tal manera que pueda realizar cualquier cambio en cualquier momento en el código.
- **Operación (Operate):** el equipo de operación se encargará del comportamiento inadecuado del sistema o de los errores que se encuentran en producción, trabajando siempre sobre un sistema sólido y estable.
- **Monitoreo (Monitor):** esta última etapa de DevOps, es una fase permanente y que se aplica a todo el ciclo completo. Aquí el equipo se encargará de definir las medidas para monitorizar y controlar el estado de salud de las aplicaciones y de su infraestructura [7] [11] [12].A continuación, en la Figura 1-3 se observa el ciclo iterativo en DevOps y la relación con las tres (3) prácticas de DevOps mencionadas anteriormente.

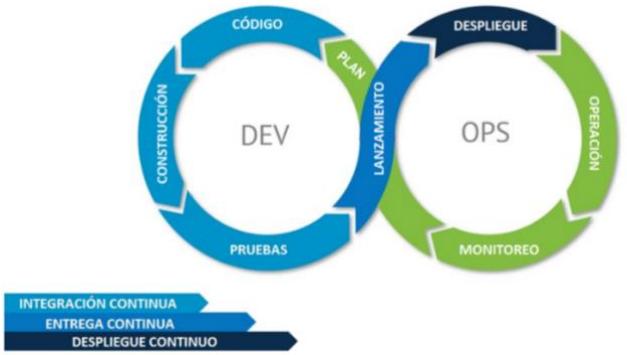



**Figura 1-3.** Ciclo iterativo DevOps. Adaptada de **[13]**

- ***Pipeline CI/CD***

Un pipeline CI/CD es un conjunto de prácticas para incorporar la automatización continua y el control permanente en todo el ciclo de vida del desarrollo de software. Desde las etapas de integración y prueba, hasta las de distribución e implementación. Entre las características de un pipeline CI/CD se tienen: integración y verificación, automatización, cultura DevOps y contenerización [14].“Un pipeline es definido por un numero de pasos lógicamente organizados, cada uno de estos pasos consiste en un número de acciones para ser llevadas a cabo” [15].

- ***Descripción de una plataforma DevOps***

El flujo de DevOps inicia desde que el desarrollador realiza un *push* (subir/cargar) de alguna característica nueva del código desde el repositorio clonado localmente (local repository) al repositorio remoto (remote repository) donde se encuentra el código fuente centralizado. Este desencadena los flujos configurados en el orquestador los cuales son ejecutados por el mismo. Luego de realizar el *push*, el código contenido en el repositorio centralizado es tomado por un gestor de compilación para construirlo, compilarlo y así generar el ejecutable o también llamado *artefacto (artifact)*. Al Artefacto generado se le realiza una verificación de calidad de código y pruebas unitarias para comprobar la calidad de este. Al ser exitosas dichas pruebas, se envía una alerta de notificación al desarrollador informando que las pruebas fueron exitosas y el *artefacto* fue construido, compilado, probado exitosamente y será almacenado y publicado en un repositorio de artefactos. Para desencadenar el despliegue del *artefacto*, se puede realizar de dos formas; la primera, manualmente conocida como entrega continua “*Continuous Delivery*” y la segunda, automática, denominada despliegue continuo “*Continuous Deployment*”. Para continuar el flujo, ya sea manual o automáticamente, se inicia la configuración automática de los entornos y estos se virtualizan a

la medida según el ambiente requerido. Al final del flujo se realiza el monitoreo continuo y la *retroalimentación* (Feedback) necesarios de la aplicación desplegada. En la Figura 1-4 observa el diagrama de componentes que actúan durante todo el ciclo DevOps, desde el *push* que realizan los desarrolladores, hasta el despliegue en su respectivo ambiente.

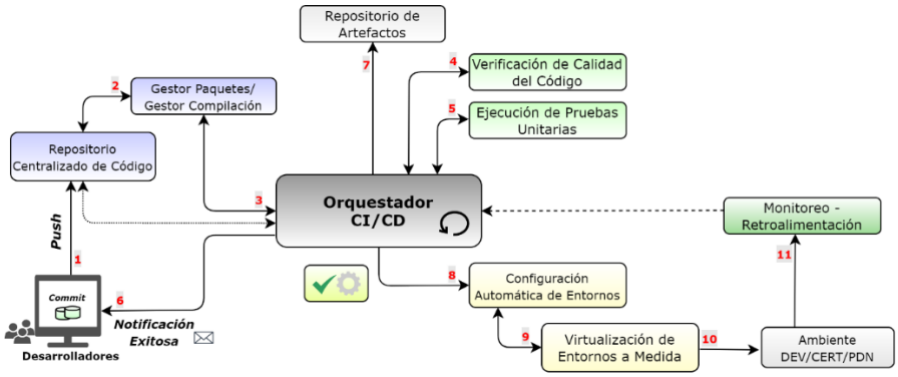



**Figura 1-4.** Descripción plataforma DevOps

Para un mayor entendimiento se hace un zoom sobre la Figura 1-4 dividiendo los componentes que hacen parte del proceso de CI o integración continua y el proceso de CD o despliegue continuo como su posible herramienta que cumple la función. La Figura 1-5 corresponde al proceso de integración continua "CI", y la Figura 1-6 corresponde al proceso de despliegue continuo "CD".

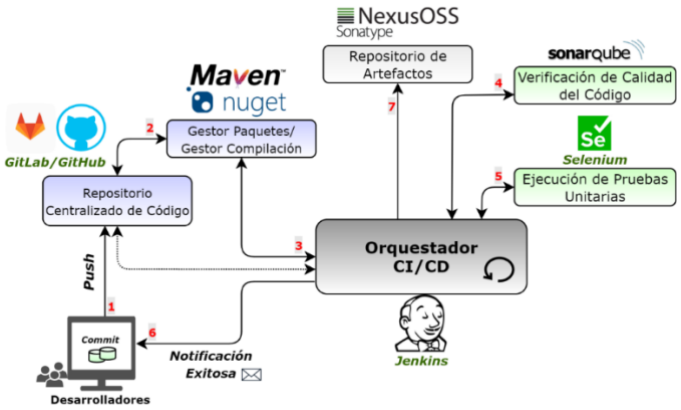



**Figura 1-5.** Componentes Integración Continua - Herramientas

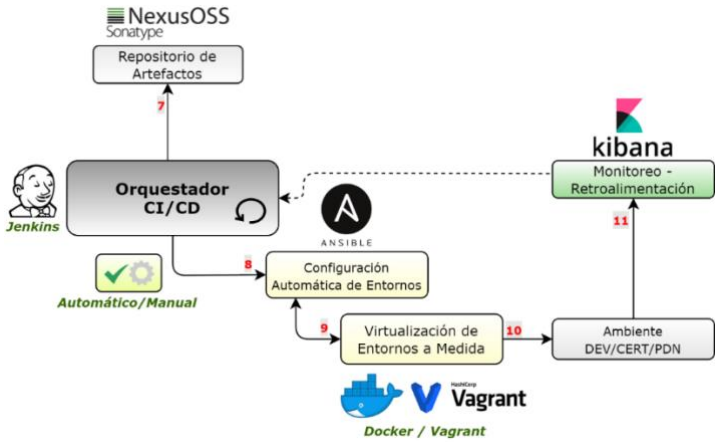



**Figura 1-6.** Componentes Despliegue Continuo - Herramientas

- ***Herramientas para implementar DevOps***

A continuación, se ilustran algunas de las herramientas utilizadas para el ciclo de integración continua "CI" Figura 1-7, proceso de despliegue continuo "CD" Figura 1-8 y en la Tabla 1-1 se muestran los integradores más conocidos.

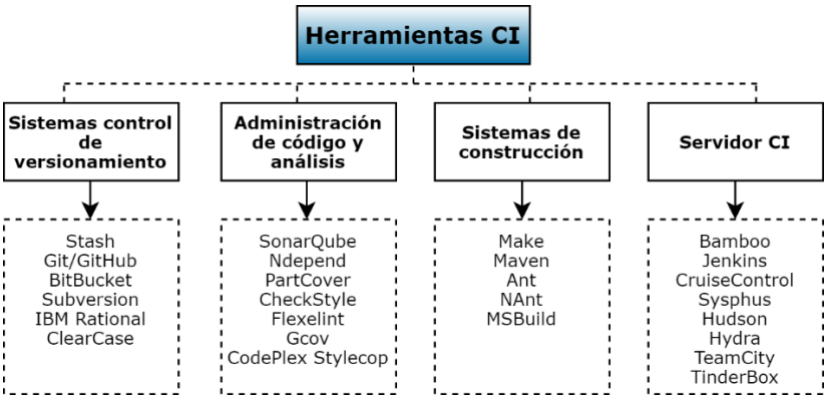



**Figura 1-7.** Herramientas DevOps para la integración continua CI. Adaptada de **[8] [16] [17] [18] [19]**

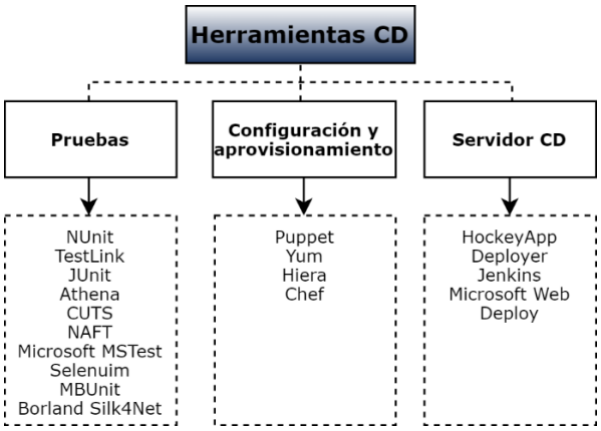



**Figura 1-8.** Herramientas DevOps para el despliegue continuo CD. Adaptada de **[8] [16] [17] [18] [19]**

**Tabla 1-1.** Integradores CI/CD. Adaptada de **[18] [19] [20] [21] [22]**

| Integradores CI/CD |
|---|
| Bamboo |
| Buildbot |
| Concourse CI |
| GOCD |
| Jenkins |
| TeamCity |
| Travis CI |

### 1.1.4. Seguridad en aplicaciones web

Se abordan algunos conceptos básicos de seguridad y luego se ahonda en la seguridad en aplicaciones web, en los cuales se mencionan diferentes temas como: estándares, normas y buenas prácticas, vulnerabilidades más comunes, herramientas para garantizar la seguridad en aplicaciones web, entre otros.

- ***Seguridad de la información - ciberseguridad***

La seguridad de la información abarca las medidas y actividades que intentan proteger los activos de información, es decir, la protección de la información o datos que tienen valor para una organización, a través de la reducción de riesgos y mitigando las amenazas posibles. Estos activos se pueden encontrar en diferentes formatos, por ejemplo, en formato digital, de forma física o en forma de ideas o conocimientos de personas que pertenecen a la organización. ***La ciberseguridad*** tiene como foco principal la protección de la información digital de los sistemas, por ello se puede considerar que la ciberseguridad está comprendida dentro de la seguridad de la información [23].

- ***Vulnerabilidad, amenaza, riesgo***

Vulnerabilidad es una debilidad o fallo en un sistema de información que pone en riesgo la seguridad de la información pudiendo permitir que un atacante pueda comprometer la integridad, disponibilidad o confidencialidad de la misma, por lo que es necesario hallarlas e intentar mitigarlas. Estas brechas pueden tener distintos orígenes, por ejemplo: fallos de diseño, malas configuraciones o carencias de buenas prácticas de seguridad [24].

Amenaza es toda acción que aprovecha una vulnerabilidad para atentar contra la seguridad de un sistema de información. Es decir, si se aprovecha dicha vulnerabilidad se podría tener un efecto negativo sobre algún elemento de nuestros sistemas [24].

El riesgo de seguridad de la información está asociado con el potencial de que las amenazas exploten las vulnerabilidades de un activo de información o grupo de activos de información y, por lo tanto, causen daños a una organización [25].

- ***Políticas de seguridad***

Las políticas de seguridad se basan en directrices y normas que permiten garantizar los tres pilares fundamentales de la seguridad de la información: confidencialidad, integridad y disponibilidad. Cuando se define una política de seguridad es importante definir el qué y el cómo del activo que se quiere proteger y así ofrecer ciertos controles para ello. Para dar cumplimiento, se desarrolla bajo una serie de procedimientos e instrucciones, los cuales se basan en un análisis previo de los riesgos a los que se está expuesto. Una política de seguridad debe estar en constante actualización y divulgada a todo el personal de la compañía, incluyendo alta gerencia [26].

- ***Seguridad en aplicaciones web***

Se nombran algunos de los estándares, normas y buenas prácticas para aplicar seguridad en las aplicaciones web, entre estos, el proyecto abierto de seguridad en aplicaciones web “OWASP”, la lista de las 10 vulnerabilidades más comunes a tener en cuenta y las diferentes herramientas para brindar seguridad. Además, se finaliza con la citación de las diferentes fuentes, bases de datos de las vulnerabilidades conocidas.

***CIS Security Benchmark – CIS Benchmark Hardening***

Estándar que provee una guía descriptiva de cómo establecer una configuración de seguridad en la infraestructura TI, incluyendo una descripción detallada y racional de las posibles vulnerabilidades potenciales y de los posibles pasos para la remediación de estas. Para construir estas guías “CIS” se tienen como referencia las organizaciones de auditoría de corte mundial, encargadas de asesorar la adopción de estándares seguros en cualquier iniciativa de gobernanza y seguridad [27].

### *El proyecto abierto de seguridad en aplicaciones web*

OWASP, Open Web Application Security Project. Es una comunidad abierta dedicada a posibilitar que las organizaciones desarrollen, adquieran y mantengan aplicaciones y APIs confiables. Para lograr esto, brinda herramientas, estándares de seguridad en aplicaciones, revisión de seguridad en aplicaciones, desarrollo de código seguro y revisiones de seguridad en código fuente [28].

### *OWASP Top 10*

El OWASP Top 10 es un documento de conocimiento estándar para desarrolladores y seguridad de aplicaciones web, donde se construye un amplio consenso sobre los riesgos de seguridad más críticos para las aplicaciones web. Al adoptar este documento, las empresas darían el primer paso hacia el desarrollo seguro de software y contarían con un soporte para minimizar los riesgos a los cuales las aplicaciones podrían estar expuestas [29]. A continuación, en la Figura 1-9 se muestran el top 10 de vulnerabilidades del año 2017 con respecto al año 2021.

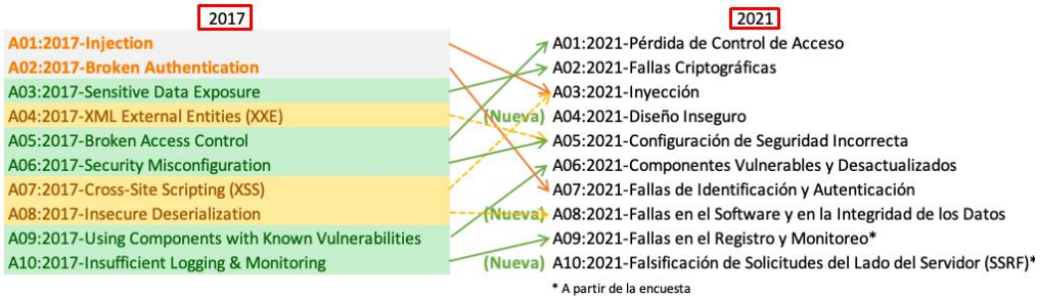



**Figura 1-9.** Comparación OWASP Top 10: 2017-2021 **[30]**

### *Estándar de verificación de seguridad en aplicaciones 3.0.1*

ASVS, Application Security Verification Standard. El estándar es una lista de requerimientos de seguridad o pruebas que pueden ser utilizados por arquitectos, desarrolladores, testers, profesionales de seguridad, para definir el nivel de seguridad de una aplicación. ASVS posibilita el trabajo en el desarrollo y seguridad en aplicaciones, además aporta a la alineación entre necesidades y servicios de seguridad, proveedores de herramientas de seguridad y consumidores. Cuenta con tres niveles de verificación de seguridad, incrementando la profundidad con respecto al nivel:

- ASVS nivel 1, dirigido a todo tipo de software.
- ASVS nivel 2, aplicaciones con datos sensibles, requieren protección.

- ASVS nivel 3, aplicaciones críticas, aplicaciones transaccionales, datos confidenciales o cualquier tipo de aplicación que requiera un tipo de confianza más avanzado [31].

***OWASP Marcos de conocimiento sobre seguridad***

SKF, Security Knowledge Frameworks. Es una base de conocimientos de seguridad que incluyen proyectos gestionables con listas de verificación y ejemplos de buenas prácticas para implementar en código bajo múltiples lenguajes de programación, además, muestra cómo se puede prevenir que se gane acceso en la aplicación y la ejecución de exploits dentro de esta; para los requerimientos de seguridad, usa el estándar ASVS, y, además, brinda una retroalimentación a los desarrolladores en cuanto a las descripciones y el cómo implementar dichos controles de seguridad. Como paso posterior, SKF valida si el desarrollador implementó los controles creados en la ASVS correctamente [32] [33].

***OWASP Guía de pruebas de seguridad web***

WSTG, Web Security Testing Guide. Es una guía completa para probar la seguridad de aplicaciones y servicios web, proporciona un marco de mejores prácticas utilizadas por las personas que realizan pruebas de penetración y por diferentes organizaciones a nivel mundial [34] [35].

A continuación, se mencionan algunas de las vulnerabilidades más comunes en las aplicaciones web:

- ***Vulnerabilidades en aplicaciones web***

Existen una gran cantidad de vulnerabilidades en aplicaciones web, en este apartado se describen las vulnerabilidades más comunes y mencionadas en el OWASP TOP 10, la cuales se deben tener en cuenta cuando se trabajan con aplicaciones web.

***Secuencia de comandos entre sitios***

XSS, Cross Site Scripting. Tipo de inyección, en el que se introducen scripts maliciosos en sitios web considerándolos benignos y confiables. Los ataques XSS ocurren cuando un atacante usa una aplicación web para enviar código malicioso, generalmente en forma de un script del lado del navegador a un usuario final diferente. Las fallas que permiten que estos ataques tengan éxito ocurren cuando la aplicación web usa la entrada de un usuario dentro de la salida que genera sin una validación previa o sin la codificación pertinente [36].

***Solicitud de sitios cruzados/Falsificación de referencias***

CRSF, Cross Site Request/Reference Forgery. Es un ataque que obliga a un usuario final ejecutar acciones no deseadas en una aplicación en la que se está autenticando actualmente. Bajo ataques como ingeniería social, un atacante puede engañar a los usuarios de una aplicación web para que ejecuten las acciones que el atacante elija. Si la víctima es un usuario normal, un ataque exitoso podría obligar al usuario a realizar un cambio de estado en las solicitudes. Si la cuenta es administrativa, CSRF puede comprometer toda la aplicación web [37].

***Inyección SQL***

SQL Injection. Primero se introduce la definición de SQL. SQL es un lenguaje declarativo de acceso a bases de datos que permite consultar, insertar y modificar la información. Su sencillez y longevidad hicieron que SQL sea el lenguaje de acceso de base de datos más utilizado al día de hoy. Gestores de bases de datos como MySQL, Oracle o SQLite, entre muchos otros, están basados en SQL. Esto ha provocado que la gran mayoría de lenguajes de programación hayan implementado librerías para trabajar con SQL, lo que ha derivado en su utilización masiva en cualquier tipo de aplicaciones [38]. SQL injection consiste en la inserción o inyección de una consulta SQL a través de los datos de entrada de un cliente a la aplicación. Si la inyección es exitosa, se podrían leer datos confidenciales en la base de datos, modificar los datos, ejecutar operaciones de administración y, en algunos casos, ejecutar comandos para el sistema operativo [39].

***Evaluación de código remoto***

Remote Code Evaluation. Vulnerabilidad que puede explotarse si la entrada del usuario se inyecta en un archivo o cadena y es ejecutado por el analizador del lenguaje de programación. Por lo general, este comportamiento es pasado por desapercibido por los desarrolladores. Una evaluación remota de código puede llevar a un compromiso total de la aplicación web vulnerable y también del servidor web. Es importante tener en cuenta que la mayoría de lenguajes de programación tienen funciones de evaluación de código [40].

***Banderas de Control de Acceso***

Access Control Flags. El control de acceso, a veces conocido como “autorización”, es la forma en la que la aplicación web otorga acceso al contenido y funciones a los usuarios. Estas comprobaciones se realizan

después de la autenticación y rigen las tareas que pueden ejecutar los usuarios autorizados. El modelo de control de acceso de una aplicación web está relacionado con el contenido y las funciones de esta. Muchos de los esquemas de control de acceso no son diseñados ni probados con anterioridad, simplemente han evolucionado junto con el sitio web. En este caso, las reglas de control de acceso se insertan en varias porciones del código. Al descubrir la falla del esquema de control de acceso defectuoso, el atacante podría ver contenido no autorizado, además, podría cambiar o eliminar contenido sin previa autorización [41].

***Pérdida de Autenticación y Gestión de Sesiones***

Sucede cuando las funciones de autenticación son implementadas incorrectamente, las cuales podrían comprometer contraseñas, token de sesiones o toma de identidad de otros usuarios [28].

***Exposición de Datos Sensibles***

Cuando las aplicaciones web y las APIs no protegen datos sensibles, puede ser aprovechado por atacantes para robar o modifigar dichos datos. Como una posible solución, se recomienta implementar métodos de crifrado en tránsito y almacenamiento [28].

***Entidad Externa de XML (XXE)***

Provocada por la mala configuración en procesadores XML. Estos son usados para revelar archivos internos mediante la URI o en plataformas no actualizadas, escanear puertos, ejecutar código y realizar ataques de denegación de servicio (DoS) [28].

***Deserialización Insegura***

Ocurre cuando una aplicación recibe objetos serializados dañidos, los cuales pueden ser manipulados por los atacantes para realizar ataques de repetición, inyecciones o elevar privilegios de ejecucición [28].

***Componentes con Vulnerabilidades Conocidas***

En muchos casos los componentes del software como: independencias, frameworks y otros módulos se ejecutan con los mismos privilegios que la aplicación. Si alguno de dichos componentes es vulnerable, el ataque puede causar pérdida de datos o tomar control de la plataforma. Las aplicaciones y API’s con componentes vulnerables conocidos pueden debilitar las defensas y permitir impactos y/o diversos ataques [28].

***Registro y Monitoreo Insuficientes***

La insuficiencia de estos, junto a la poca respuesta ante un incidente permite al atacante mantener el ataque en el tiempo, realizar pivotes hacia otro sistemas, manupilar, extraer y destruir datos [28]. Es de gran importancia indicar que las vulnerabilidades anteriormente mencionandas ya se encuentran caracterizadas, clasificadas y con posibles soluciones para cerrar dicha brecha. La siguiente vulnerabilidad no se encuentra en la lista "Top 10" de OWASP, pero es de gran importancia referirla.

***Vulnerabilidad de día cero***

Zero Day Vulnerability. Nueva vulnerabilidad que aparece y aún no se le han creado parches o revisiones. No existe revisión alguna para mitigar el aprovechamiento de dicha vulnerabilidad [42].

- ***Herramientas OWASP***

Algunas de las herramientas recomendadas por OWASP para ofrecer la seguridad en aplicaciones web.

***DefectDojo***

Programa de seguridad y herramienta de administración/gestión de vulnerabilidades. Permite administrar el programa de seguridad de la aplicación, mantener la información de productos y aplicaciones, realizar escaneos, clasificar vulnerabilidades y consolidar los hallazgos [43] [44].

***SonarQube***

Herramienta de revisión automática de código para detectar errores, vulnerabilidades en el código. Inspección continua del código e integración con el flujo de trabajo existente [44] [45].

***Find Security Bugs***

Herramienta de auditoría de seguridad de aplicaciones web Java bajo el complemento SpotBugs. GoSecure es el encargado del desarrollo desde el año 2016. El soporte incluye el desarrollo de nuevos detectores y la investigación de nuevas clases de vulnerabilidades [44] [46].

***LGTM***

Plataforma de análisis que verifica automáticamente su código en busca de CVE y vulnerabilidades. Se puede realizar una combinación con ciencia de datos en la búsqueda de código para dar resultados más

relevantes, ofrece además información de una comunidad de investigadores de seguridad para así ayudar a los desarrolladores a enviar código seguro [44] [47].

***OSS Index***

Cátalogo gratuito de componentes de código abierto y herramientas de escaneo para ayudar a los desarrolladores a identificar vulnerabilidades, comprender los riesgos y mantener su desarrollo seguro [44] [48].

***Snyk***

En la versión open source, snyk ayuda a encontrar y corregir automáticamente vulnerabilidades y violaciones de licencia en sus dependencias de código abierto. Este es diseñado para colaborar a los desarrolladores y que puedan integrarse en sus flujos de trabajo existentes, proporcionando soluciones automatizadas y conocimientos de seguridad procesables [44] [49].

- ***Bases de datos de vulnerabilidades para su análisis***

Bases de datos, repositorios y lista de entrada donde se exponen y se clasifican las diferentes vulnerabilidades que son identificadas día a día.

***Vulnerabilidades y exposiciones comunes***

CVE, Common Vulnerabilities and Exposures. Lista de entradas de vulnerabilidades compuestas por: un número de identificación, descripción y al menos una referencia pública para vulnerabilidades de ciberseguridad conocidas públicamente [50].

***Enumeración de debilidades comunes***

CWE, Common Weakness Enumeration. Lista desarrollada por la comunidad de desarrolladores de tipos de debilidades de software y hardware. Lenguaje común y un punto de partida para medir las herramientas de seguridad y, además, utilizada como base para los esfuerzos de identificación, mitigación y prevención de debilidades [51].

***Sistema de puntaje para vulnerabilidades comunes***

CVSS, Common Vulnerability Scoring System. “Framework abierto utilizado para establecer métricas para la comunicación de las características, impacto y severidad de vulnerabilidad que afectan a elementos del entorno de seguridad TI” [52].

***Base de datos nacional de vulnerabilidades***

NVD, National Vulnerability Database. Repositorio del gobierno de EEUU de estándares basados en administración de vulnerabilidades representados mediante el protocolo de automatización de contenido de seguridad (SCAP, Security Content Automation Protocol). Estos datos permiten la automatización de la administración de vulnerabilidades, medición de seguridad y el cumplimiento. NVD incluye bases de datos de referencias de listas de verificación de seguridad, fallas de software relacionadas con la seguridad, configuraciones incorrectas, nombres de productos y métricas e impacto [53].

***VulDB***

Es considerada la base de datos de vulnerabilidades número uno (1) a nivel mundial [54]. La comunidad está compuesta por expertos que se han dado al trabajo de documentar las vulnerabilidades diariamente. La base de datos contiene más de 110000 entradas, en esta se incluyen las entradas de otras bases de datos, como CWE y CVE. A diferencia de otras bases de datos de vulnerabilidades, VulDB describe, categoriza, califica la vulnerabilidad con la evaluación del sistema de puntaje para vulnerabilidades comunes (CVSS, Common Vulnerability Scoring System) e incluye información acerca de la vulnerabilidad [18].

***CIRCL CVE Search***

Interfaz accesible vía web mediante el consumo de una API. Interfaz para buscar públicamente la información conocida de vulnerabilidades de seguridad en software y hardware junto con sus correspondientes exposiciones. Entre las bases de datos más comunes a las cuales se realizan las consultas, tenemos: base de datos nacional (NIST), plataforma común de enumeración (CPE), enumeración de debilidades comuniones (CWE), entre otras [55].

### 1.1.5. Seguridad en DevOps - SecDevOps

Cuando se habla de seguridad en DevOps implica pensar desde el principio tanto en la seguridad de las aplicaciones y la infraestructura, como de automatizar algunas estrategias de seguridad para evitar lentificar el flujo de trabajo de DevOps [56]. La premisa principal de SecDevOps es que todo agente involucrado en el ciclo de vida del software es responsable de la seguridad, en esencia, es la colaboración entre los equipos de desarrolladores, operaciones y seguridad.

En la Figura 1-10 se observan las estrategias de seguridad recomendadas durante el ciclo de DevOps.


**Figura 1-10.** Estrategias de seguridad durante el ciclo iterativo de DevOps

Es de gran importancia introducir el concepto de romper el "Build", dado que nos ofrece un control y cumplimiento de políticas de seguridad establecidas.

- ***Romper la construcción (RB)***

RB, que es la abreviatura de Romper el Build, este término hace referencia a la acción a tomar para detener el proceso dentro del ciclo de vida del desarrollo del software cuando se encuentre un fallo que por lo general hace referencia a la calidad de dicho software, pero en esta propuesta de SecDevOps se toma también como una violación en los umbrales de seguridad determinados por la política, lo cual indica que el código no promocionará a la siguiente fase.

"Algo muy importante es saber si el código que se quiere promover es seguro según su construcción o si definitivamente, de manera automática, "rompemos el Build" para que los hallazgos sean solucionados antes de incluir sus cambios en ambientes pre-productivos o, en el lenguaje GIT, antes de hacer "merge" con una rama principal. Esto se puede condicionar con estrategias como: PullRequest (PR) o MergeRequest (MR) con validaciones atadas a los resultados, según las herramientas utilizadas" [57].

- ***Política de seguridad para la evaluación de Romper el Build***

Basados en la definición general de política de seguridad presentada en este apartado. La política de seguridad que se va a emplear en esta investigación se fundamenta en dos (2) elementos: el primero es el punto donde se desea evaluar el RB; para esta propuesta de seguridad, los lugares son al final de cada proceso de CI y de CD. Y la ruptura está definida por la cantidad mínima de vulnerabilidades que debe

cumplir como referencia establecida para los riesgos de seguridad asociados al aplicativo a desarrollar y que son definidos en la fase del precommit.

- ***Métodos o estrategias de seguridad en la integración continua (CI)***

A continuación, se mencionan algunas de las estrategias y métodos usados para implementar seguridad en la integración continua durante el proceso de DevOps:

***Pre-commit***

Son actividades usadas para encontrar y corregir posibles errores comunes antes de la verificación de los cambios por parte de los repositorios del código fuente [58]. "Git Hook, son scripts de shell que se ejecutan automáticamente antes o después de que git ejecute un comando importante como un *commit* o un *push"* [59]. A continuación, se mencionan diferentes estrategias que cumplen la función de verificar el código:

- Modelamiento de amenazas y mapeo de ataques.
- Historias de seguridad y privacidad.
- Complementos de seguridad IDE (Plugins IDE).
- Estándares de código seguro.
- Revisiones manuales y por pares.
- Detección y eliminación de información sensible en los mensajes enviados al repositorio [58].

***Gestión de secretos***

Management Secrets. "Refiere a las herramientas y métodos para la gestión de credenciales de autenticación digitales (secretos), incluyendo contraseñas, llaves, APIs, tokens para el uso de aplicaciones, servicios, cuentas privilegiadas y otras partes sensibles del entorno de TI" [60].

***Modelado de amenazas***

Técnica de comprobación usada con el objetivo de ayudar a identificar y a planificar de la forma correcta el cómo se mitigarían las amenazas de una aplicación mediante un enfoque orientado al análisis de gestión de riesgos, además, de la implementación de medidas y controles que apoyen a mejorar la seguridad. El modelado debe contar y seguir con una metodología que esté en constante evolución, para así mitigar la

mayor cantidad de vulnerabilidades en fases tempraneras del ciclo de vida del desarrollo de software. Dicho modelo debe contar con las siguientes características:

1. Identificar las amenazas potenciales.
2. Identificar las vulnerabilidades eliminadas o contrarrestadas.
3. Proporcionar información relevante acerca de los controles más eficaces para contrarrestar un posible ataque.
4. Comunicación a la alta gerencia sobre los riesgos tecnológicos en términos de impacto de negocio.
5. Facilitar la comunicación y promover la importancia de la seguridad.
6. Simplificar la actualización mediante de componentes reutilizables.

Algunas de las metodologías que se pueden usar en el modelado de amenazas son: Ace Threat Analysis and Modeling de Microsoft, CORAS (Consultative Objetive Risk Analysis System), Trike, PTA (Practical Threat Analysis), y STRIDE [61].

***Controles proactivos OWASP***

OWASP, Proactive Controls. Los controles proactivos recomendado por OWASP describen los controles de seguridad y las categorías más importantes que se deben tener en cuenta para diseñar y desarrollar un software, los cuales deberían ser incluidos al 100% en cada proyecto. Los diez (10) controles se mencionan a continuación:

1. C1: Definición de requerimientos de seguridad.
2. C2: Aprovechamiento de marcos de trabajo de seguridad y librerías.
3. C3: Asegurar el acceso a la base de datos.
4. C4: Codificar y escapar información.
5. C5: Validar todas las entradas.
6. C6: Implementar identidad digital.
7. C7: Forzar los controles de acceso.
8. C8: Proteger la información en todo lugar.

9. C9: Implementar monitoreo y registro de seguridad.
10. C10: Manejar todos los errores y excepciones [62].

***Pruebas de seguridad de aplicaciones estáticas***

SAST, Static Application Security Testing. "Conjunto de tecnologías diseñadas para analizar el código fuente de aplicaciones, el código de bytes y los binarios para las condiciones de codificación y diseño que son indicativas de vulnerabilidades de seguridad" [63]. Con esta estrategia se analiza el código de una aplicación siguiendo una serie de reglas que buscan patrones y flujos en el propio código fuente sin necesidad de compilar el código, siguen varios estándares de análisis de vulnerabilidades tales como CWE, CVE, VulBD, entre otros. Este tipo de estrategia permite integrarse en los sistemas o prácticas de DevOps de integración continua, permitiendo monitorizar el código y determinar las vulnerabilidades que se van produciendo [19].

***Análisis de dependencias***

Al desarrollar una aplicación se emplean librerías y dependencias que aportan funcionalidades al aplicativo, muchas de estas dependencias son tercerizadas y no conocemos su origen. Además, en la actualidad hay gran colaboración con una gran de comunidades de desarrolladores de código abierto, muchos de estos desarrolladores no cuentan con conocimientos avanzados de seguridad y podrían causar muchas brechas de seguridad, por ello, es importante el uso de un análisis de dependencias, que por norma general analizan los gestores de dependencias empleados por el software para la compilación [19].

***Análisis de composición de software***

SCA, Software Composition Analysis. Proceso de automatizar la visibilidad del uso de software de código abierto (OSS, Open Source Software), binarios y dependencias con el fin de realización una gestión sobe los riesgos, la seguridad y el cumplimiento de seguridad de las licencias usadas en el desarrollo del software. Al hacer uso del código abierto (OS, Open Source), es necesario indagar los componentes usados ya que el riesgo crece exponencialmente debido a las vulnerabilidades que se heredan del uso de código abierto. Como consecuencia al uso de SCA, los equipos de seguridad y desarrollo pueden: crear una lista de materiales (BOM, Bill of Materials) precisa para las aplicaciones, realizar un rastreo de todas las fuentes abiertas, configurar y dar por cumplimento las políticas establecidas, habilitar un monitoreo continuo y proactivo e integrar sin problemas el escaneo de código abierto en el entorno de construcción [64].

- ***Métodos o estrategias de seguridad en el despliegue continuo (CD)***

También cabe mencionar algunas de las estrategias y métodos usados para implementar seguridad en el despliegue continuo durante el proceso de DevOps:

***Evaluación de vulnerabilidades***

VA, Vulnerability Assessment. Revisión sistemática de las debilidades de seguridad en un sistema de información. Evalúa si un sistema es susceptible a una vulnerabilidad conocida, categoriza la vulnerabilidad asignado niveles de severidad y recomienda planes para la remediación o mitigación de esta. Algunas de las amenazas que pueden ser prevenidas: SQL injection, XSS, ataques de inyección de código, escalamiento de privilegios, entre otros [65].

***Gestión de vulnerabilidades***

Vulnerability Management. "Proceso continuo que incluye la detección proactiva de activos, monitoreo continuo, mitigación, y las tácticas de defensa que son necesarias para proteger la superficie de ataque del entorno TI" [66].

***Análisis de vulnerabilidades en imágenes Docker***

Análisis de seguridad a nivel binario de las imágenes del repositorio antes de que sean liberadas. Para esto es necesario realizar un escaneo de seguridad docker (DSS, Docker Security Scanning), servicio que se dispara cuando una imagen docker se sube a un repositorio, se recibe la imagen y se descomponen en sus respectivos componentes, luego de realizar la descomposición hace una relación con las bases de datos de vulnerabilidades conocidas para verificar finalmente los componentes a nivel binario para validar el contenido [67]. En la Figura 1-11 se puede visualizar el funcionamiento del servicio DSS.

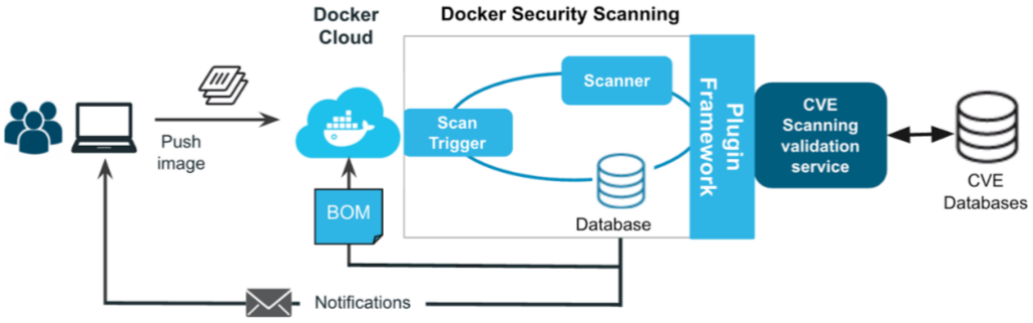



**Figura 1-11.** Funcionamiento servicio DSS. Tomada de **[67]**

***Pruebas de penetración***

Penetration Testing. Pruebas ofensivas para medir el nivel de protección de los mecanismos de defensa existentes en el entorno analizado. El objetivo de estas pruebas es verificar bajo situaciones extremas cuál es el comportamiento de los mecanismos de defensa, específicamente, se busca detectar vulnerabilidades en los mismos. Además, se identifican aquellas faltas de controles y las brechas que pueden existir entre la información crítica y los controles existentes. Las fases utilizadas para la prueba son: fase de reconocimiento, escaneo, enumeración, acceso, mantenimiento de acceso [68].

***El fortalecimiento de sistemas***

System Hardening. Colección de herramientas, técnicas y mejores prácticas para reducir la vulnerabilidad en aplicaciones de tecnología, sistemas, infraestructura, firmware y otras áreas. El objetivo del fortalecimiento de los sistemas es reducir el riesgo de seguridad mediante la eliminación de posibles vectores de ataque y la condensación de la superficie de ataque del sistema. Al eliminar programas superfluos, funciones de cuentas, aplicaciones, puertos, permisos, acceso, etc., los atacantes y el malware tienen menos oportunidades de establecerse en su ecosistema de TI [69].

***Pruebas de seguridad de aplicaciones dinámicas***

DAST, Dynamic Application Security Testing. Test diseñados para detectar vulnerabilidades de seguridad de una aplicación en su estado de ejecución. La mayoría de las soluciones DAST prueban solo las interfaces HTTP y HTML expuestas en aplicaciones habilitadas para WEB [70]. Estas pruebas se centran en la búsqueda de vulnerabilidades en tiempo real mientras la aplicación se está ejecutando, el objetivo de las pruebas DAST es detectar vulnerabilidades que no se detectaron en las fases anteriores [16]. Además, DAST es conocida como una “prueba de caja negra” debido a que esta es realizada sin previa visualización sobre el código fuente o arquitectura de la aplicación, usa técnicas similares que un atacante podría usar para encontrar debilidades potenciales [71].

- ***Herramientas para implementar SecDevOps***

**Tabla 1-2.** Herramientas para implementar seguridad en el ciclo de integración continua. Adaptada de **[19] [58] [72] [22]**

| **Herramientas CI – SecDevOps** | |
|---|---|
| OWASP ThreatDragon | FindSecurityBugs |
| OWASP ASVS | ESLint |
| DevSkin | JUnit |
| SonarLint | Puppet-lint |
| Git-Secrets | OWASP Dependency Check |
| OWASP Practive Controls | Node Security Platform |
| GitLab Merge Request | CIS Benchmarks |
| SonarQube | BitBucket |
| PMD | OSSIndex |
| Black Duck | |

Se realiza un estudio de algunas herramientas, de todas las características, para implementar SecDevOps tanto en el ciclo de integración continua y en el ciclo de despliegue continuo. Para visualizar el estudio ver la Tabla 1-2 y .

Tabla 1-3.

**Tabla 1-3.** Herramientas para implementar seguridad en el ciclo de despliegue continuo. Adaptada de **[16] [18] [19] [73] [74] [75]**

| **Herramientas CD – SecDevOps** | |
|---|---|
| Clair | Jenkins CI Image Vulnerability Scan |
| Dagda | CIS Benchmark |
| Anchore Engine | Nessus |
| OWASP Zed Attack Proxy (ZAP) | OpenVAS |
| Anchore Container Image Scanner | Qualys |
| Snyk | Nikto2 |
| RunDeck | Acunetix |
| Docker Hub Image Security Scan | Nikto |

## 1.2. Estado del arte

Aladrén García Julio [74], ahonda en los mecanismos para asegurar ambientes basados en docker y kubernetes, con la intención de definir un baseline (punto de partida, a partir del cual se podrá construir un sistema desde bases y definiciones) basado en Anchore (Anchore Container Image Scanner Plugin), el cual podrá ser integrado a distintos software de automatización en las prácticas de integración continua orientados a entornos de DevSecOps, en los que sea posible realizar un análisis de vulnerabilidades, esto con el objetivo de asegurar los entornos en los que se haga uso de ellos. Para esto se hace el uso de Anchore con Jenkins para realizar un análisis de vulnerabilidades con carácter preventivo y así brindar recomendaciones de seguridad durante la fase de pruebas del proyecto, además se realiza una segmentación entre ambientes para un entorno laboral, los cuales se componen de entornos de pruebas, administración y producción.

En la investigación "Docker Container Security in Cloud Computing" [76] se expone que debido a la migración de servicio y de infraestructura hacia la nube, se hace necesario la validación de imágenes de docker que son tomadas de varios repositorios tanto públicos y privados. Específicamente se describe un sistema basado en integración continua (CI, Continuos Integration), y despliegue continuo (CD) donde se valida la seguridad de las imágenes de docker a través de un ciclo de vida de desarrollo de software. Se hizo un estudio de la tecnología de contenedores, buenas prácticas de seguridad en imágenes de docker, escaneo de virus y un análisis dinámico para evaluar la seguridad de los contenedores docker, usando herramientas como CoreOS Clair, Anchore Engine, entre otras. Como resultado se implementó una API para automatizar y capturar información útil acerca de imágenes docker específicas para identificar bugs y contenidos maliciosos. Esta implementación de la API tiene dos propósitos: en primer lugar, es una herramienta de investigación para que los desarrolladores analicen imágenes públicas y, en segundo lugar, sirve como una segunda capa del mecanismo de seguridad de varias capas.

Por otra parte, Caño Quintero José [16] propone la automatización de controles de seguridad en ciertas fases del desarrollo del software como: (1) Las pruebas estáticas de seguridad (SAST) mediante controles de calidad de software y la comprobación de vulnerabilidades en las dependencias, (2) pruebas dinámicas de seguridad (DAST) mediante el análisis dinámico de una aplicación WEB y (3) a nivel de infraestructura se realiza un análisis de vulnerabilidades CVE en las imágenes de Docker. El desarrollo de la automatización se hace bajo tres proyectos independientes en Jenkins y el uso extenso de Docker. Concluyendo que la

realización del trabajo demuestra que es posible automatizar los controles de seguridad durante diferentes etapas del desarrollo software.

En esta misma línea Koopman Michael [77], ofrece un "framework" para detectar y prevenir las vulnerabilidades de seguridad en los pipelines de integración continua (CI, Continuous Integration) y entrega continua (CDe, Continuous Delivery), en el contexto de una gran compañía de consultoría. Para levantar la información actual se hacen ciertas investigaciones con expertos de la compañía sobre cómo el CI/CD se utiliza dentro de la empresa y cómo se encuentra construido. La metodología del diseño de la ciencia de Wieringa fue utilizada para el desarrollo de la investigación, la cual se concibió alrededor de tres (3) preguntas: 1) ¿A qué tipos de riesgos están expuestas las empresas que utilizan una línea de producción gestionada? 2) ¿Qué tipos de prácticas existen para mitigar cada tipo de riesgo? 3) ¿Cómo deberían estos riesgos ser mitigador para cada nivel de riesgo? Como resultado, se obtiene la línea base para que la compañía pueda detectar y prevenir las vulnerabilidades de seguridad en su plataforma

De igual manera, en la investigación "Seguridad en el ciclo de vida del desarrollo del software DevSecOps" [19] se propone un diseño de una solución que permita incluir la seguridad en todas las fases del ciclo de vida del desarrollo del software, apoyándose en la metodología DevOps para lograr el acercamiento entre equipos y los pilares de automatización e integración continua. Dicha solución plantea la definición de requisitos y políticas de seguridad para la fase de diseño y planificación del proyecto, así como la especificación de un pipeline seguro sobre herramientas open source. En el trabajo se demuestra la importancia de añadir la seguridad en todas las fases y se ha eliminado la idea de que seguridad y desarrollo son incompatibles, al igual de que la seguridad solo debe incluirse al final del ciclo. Como resultado, se logró realizar un diseño e implementación en el ciclo de vida del desarrollo de software gracias a la cultura SecDevOps. En este se obtiene una trazabilidad desde el desarrollo del software, donde se definen los requisitos de seguridad desde la fase de diseño, hasta el despliegue del pipeline 100% automático para realizar el checkout del código, compilación y despliegue, así como la ejecución de todas las pruebas de seguridad.

Dullman, Paule y Van Hoorn [22] identifican cuáles vulnerabilidades están presentes en los pipelines de entrega continua (CDe, Continuos Delivery) de la industria y cómo pueden ser detectadas. Como resultados del caso de estudio se incluyen: (1) una encuesta a los equipos ágiles de la compañía sobre la conciencia de seguridad en el CI/CD, (2) análisis y abstracción de los dos pipelines CDe (Alpha – Beta) usados como caso de estudio y (3) el análisis de vulnerabilidades basado en el pipeline CDe deducido para

identificar vulnerabilidades. Dicho caso de estudio arroja que los equipos no tienen una sólida formación en seguridad, pero a pesar de esto, se tiene una conciencia de los atributos de seguridad en general. Además, bajo el modelo usado para el análisis de amenazas “STRIDE” y usando los estándares propuestos por NIST y OWASP para la clasificación de la severidad de los riesgos, se identificaron 22 vulnerabilidades

En la investigación “Securing DevOps – Detection of vulnerabilities in CDe pipelines” [18], Paule Christina estudió el nivel de seguridad de los pipelines de entrega continua (CDe, Continuous Delivery) basado en el análisis e identificación de vulnerabilidades en estos pipelines; para esto, se hace uso de una encuesta en una compañía de 19 empleados, donde se identifica que algunos realizan ciertas tareas sobre los pipelines CDe, pero raramente estos cuentan con habilidades y conocimientos de seguridad. Para la detección de vulnerabilidades en los pipelines CDe se hizo uso del método de amenazas "STRIDE". Para las pruebas, se hace uso de dos pipelines CDe (CDe_A - CDe_B); el resultado muestra tipos de conexión desencriptadas entre pipelines CDe y componentes con acceso restringido que son posibles vulnerabilidades para el pipeline CDe, se compone de un set de herramientas que son necesarias para la continua detección automática de las vulnerabilidades teóricamente exploradas. Los resultados de esta tesis muestran que hay existencia de por lo menos una herramienta de dirección o mitigaciones de vulnerabilidades en cada pipeline Cde, los cuales son usados en la compañía seleccionada. Ambos pipelines CDe incluyen vulnerabilidades que tienen altos potenciales de riesgos. El resultado de la encuesta resume las posibles vulnerabilidades en los pipelines CDe:

1. Errores humanos (empleados internos).
2. Conexiones no encriptadas entre componentes de los pipelines CDe.
3. Ambientes inseguros de los componentes del pipeline CDe.
4. Ninguna o pocas restricciones de acceso.
5. Uso de versiones vulnerables de los componentes del pipeline CDe.
6. Malas configuraciones sobre los pipelines CDe.
7. Commits de código, scripts de pipelines CDe, imágenes Docker, artefactos, todos con vulnerabilidades.
8. No revisión de cambios sobre los pipelines CDe [18].

A continuación se muestra la estructura de los pipelines usados en los casos de uso, pipeline CDe_A Figura 1-12, pipeline CDe_B, Figura 1-13.

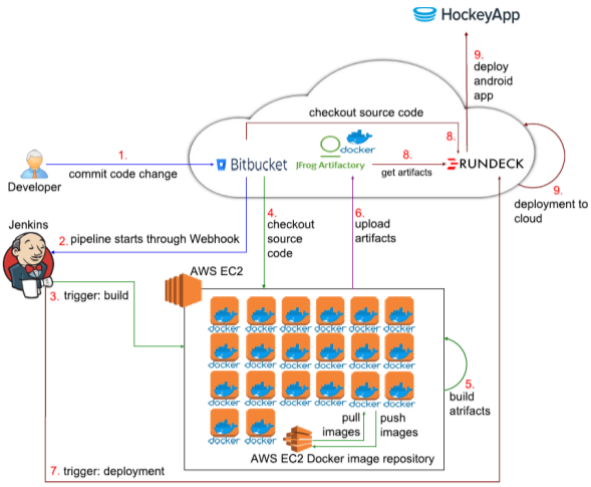



**Figura 1-12.** Estructura pipelines CDe – A. Tomado de **[18]**

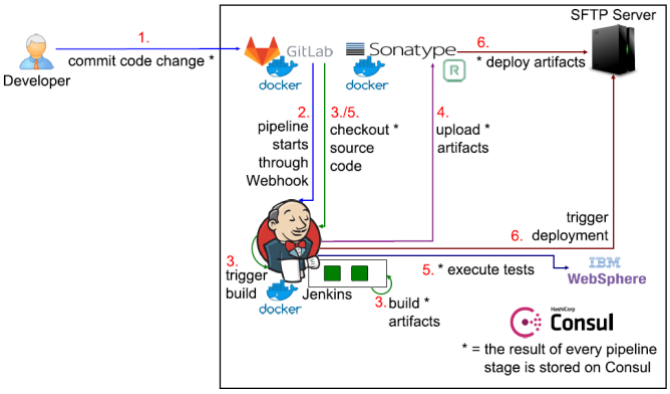



**Figura 1-13.** Estructura pipelines CDe – B. Tomado de **[18]**

Dentro de esta tesis de maestría [73] se traza el objetivo de investigar acerca de la contenerización con Docker, comprender su tecnología, funcionamiento y realizar un estudio basado en las mejores prácticas de seguridad aplicables en el universo de Docker, así como soluciones que permitan incrementar la seguridad en contenedores. Esta etapa entra en el mundo de Docker en donde normalmente va de la mano con la cultura DevOps, el análisis de vulnerabilidades de imágenes Docker, la cual cae en la primera etapa de la construcción del despliegue, sería el único análisis estático que caería dentro de las etapas incluidas dentro de los pipelines de entrega (CI, continuous integration) y despliegue continuo (CD, Continuous Deployment), surgiendo como nuevo requisito a nivel de seguridad de las imágenes de Docker como solución de empaquetado para poder gestionar, *a posteriori,* la ejecución del aplicativo contenido en dichas imágenes como contenedores en los diferentes entornos. Para la verificación de las buenas prácticas de seguridad se usó la herramienta *Docker Bench Security*, la cual está basada en un script que valida que las buenas prácticas sí están siendo implementas en las imágenes de docker antes de ser

desplegadas. Las buenas prácticas están inspiradas en el *CIS Docker Community Edition Benchmark*. Además, se tiene en cuenta otra herramienta, *anchore*, herramienta que tiene el mismo propósito de análisis estático de vulnerabilidades sobre imágenes docker almacenadas tanto en repositorios públicos, como privados.

Los autores de la investigación [78], proponen un framework DevOps para la adaptación de seguridad que permita a los equipos de desarrollo y operaciones colaborar y direccionar las vulnerabilidades de seguridad. El framework propuesto abarca las diferentes fases del software (desarrollo, operaciones, mantenimiento), además, se consideran otros factores como rendimiento, costo y funcionalidad, donde luego se decide para la adaptación de seguridad. Se muestra una aproximación en una herramienta prototipo donde se enseña cómo los equipos deben trabajar juntos para abordar las preocupaciones de seguridad. Para direccionar el reto propuesto, el framework propuesto para la adaptación de seguridad en tiempo de ejecución de sistemas de software web. El framework sigue la aproximación de DevOps propuesta por Michael Hüttermann en 2012 "DevOps for developers". Los componentes de las operaciones del framework están basadas en la arquitectura MAPE-K, para sistemas auto adaptivos, con los siguientes componentes:

1. Un sistema de monitoreo extendido. Métricas de rendimiento y utilización de los recursos, además de la inclusión de parámetros de seguridad.
2. Un motor de análisis híbrido con un componente para identificar las vulnerabilidades de seguridad en el tiempo de desarrollo y un análisis dinámico para correlacionar la información estática de desarrollo con información del tiempo de ejecución, identificando cuáles son las actuales amenazas y la evaluación de su respectivo impacto.
3. Un sistema de apoyo a la toma de decisiones como planificador de nuestro sistema MAPE, en el cual emplea la teoría de juegos y decide la acción apropiada para defender contra las amenazas.
4. Un motor de ejecución, el cual aplica acciones de adaptación planificadas en el sistema o su infraestructura.

La arquitectura está dividida en dos fases (ver Figura 1-14), la primera, la fase del desarrollo que incluye un análisis estático y tiene acceso a los componentes estáticos del sistema, tales como código fuente y políticas de seguridad. La segunda, la fase de operaciones, incluye el componente de administración dinámica del framework basado en un loop MAPE-K.

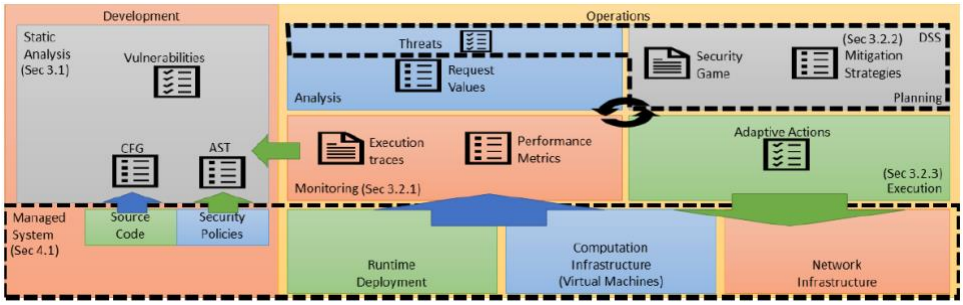



**Figura 1-14.** Arquitectura y módulos del framework de seguridad para DevOps. Tomado de **[78]**

De esta misma manera, la investigación "Security Support in Continuous Deployment Pipeline" [79], propone un diseño del pipeline deployment seguro utilizando tácticas de seguridad. Se ha demostrado la eficiencia de cinco (5) tácticas de seguridad en el diseño de un pipeline seguro experimentando en dos CDP's (Continuous Deployment Pipeline). Uno incorpora tácticas de seguridad, mientras el otro no las incorpora. Ambos CDPs han sido analizados cuantitativamente y cualitativamente. Se definen los componentes principales del CDP, como los riesgos de seguridad en los componentes claves del CDP (ver Tabla 1-4) y, además, se recomiendan (5) tácticas de seguridad en varios componentes del CDP:

1. Aseguramiento del repositorio a través del acceso controlado para controlar quién puede realizar un "commit" en ciertas ramas del repositorio.
2. Conexión segura al servidor principal mediante del uso de una llave privada sobre shell segura (SSH).
3. Uso de roles en el servidor principal para el control de acceso mediante el aprovechamiento del recurso de AWS (IAM, Identity and Access Management).
4. Configuración del servidor CI para iniciar una máquina virtual (VM, Virtual Machine) con un estado claro, aprovechando el plugin de Jenkins para las VM.
5. Uso del plugin de Jenkins para asignar roles en el servidor CI y así controlar quién puede crear, modificar y eliminar pipelines

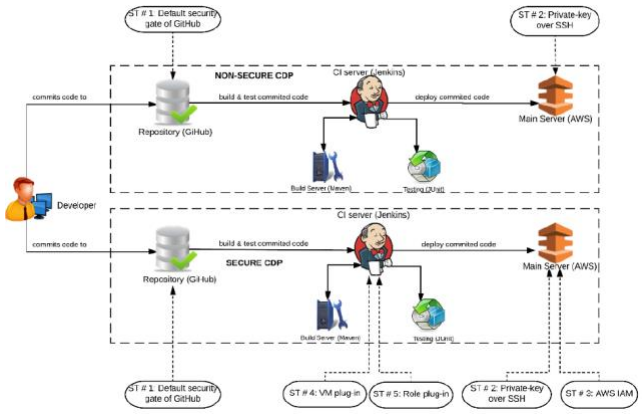



**Figura 1-15.** CDP seguro y no seguro con las tácticas de seguridad propuestas. Tomada de **[79]**

Las primeras dos tácticas son incorporadas en ambos CDP'S (seguro y no seguro), las otras tres (3) tácticas se incorporan solo en el CDP seguro (ver Figura 1-15).

**Tabla 1-4.** Componentes claves y riesgos potenciales en un CDP. Adaptada de **[79]**

| Componente | Herramienta | Versión | Riesgo de seguridad |
|---|---|---|---|
| Repositorio | GitHub | 1.9.1 | *Acceso no controlado |
| Servidor Principal | AWS | N/A | *Mecanismo de autenticación pobre<br>*Acceso no controlado |
| Servidor CI | Jenkins | 1.656 | *Inicio del proceso de construcción con un estado previo infectado<br>*Acceso no controlado |
| Pruebas | JUnit | 4.11 | |
| Servidor de construcción | Maven | 2.21 | |
| Servidor Web | Tomcat | 7.0.52.0 | |

Para el análisis cualitativo se usa el caso de garantía con notación de estructuración de objetivos (GSN, Goal Structuring Notation). El caso de garantía es una técnica de prueba donde la evidencia es organizada en un argumento para mostrar a una cierta parte interesada.

Para la evaluación de la efectividad de las cinco (5) tácticas de seguridad, se realizaron dos (2) pruebas de escaneo. Dichas pruebas lanzan varios tipos de ataques de una aplicación para encontrar vulnerabilidades y evaluar el nivel de seguridad de la aplicación. El primer escaneo es el escaneo "Qualys OWASP", el cual es normalmente practicado en entornos de aplicaciones web y está basado en estándares de seguridad por OWASP. La segunda herramienta de escaneo es OWASP Zed Attack Proxy, ZAP, usada para encontrar vulnerabilidades en aplicaciones web [79].

Partiendo de que en la metodología de software "SDLC", la seguridad no es tenida en cuenta en una prioridad alta y, además, se aplica al final del ciclo de vida del desarrollo de software, se propone la

realización de pruebas en aplicaciones web antes de ser liberadas a producción, esto con el fin de minimizar los riesgos de exposición. La autora Gómez Zafra Gina [17] por consiguiente, propone el análisis de herramientas para el testeo de aplicaciones web y desarrollo de las metodologías sobre una aplicación con HP fortify. Se mencionan algunas herramientas para pruebas de seguridad de aplicaciones estáticas (SAST) y pruebas de seguridad de aplicaciones dinámicas (DAST), que pueden ser útiles en la fase de implementación y prueba, respectivamente, y tienen el beneficio de garantizar que la seguridad se encuentre en el proceso. Para la prueba, se hace uso de una aplicación de prueba WebGoat. Además, se recomiendan los puntos donde se podría aplicar seguridad durante el ciclo de vida del desarrollo Figura 1-16, y la importancia de evaluar la prioridad de estos.

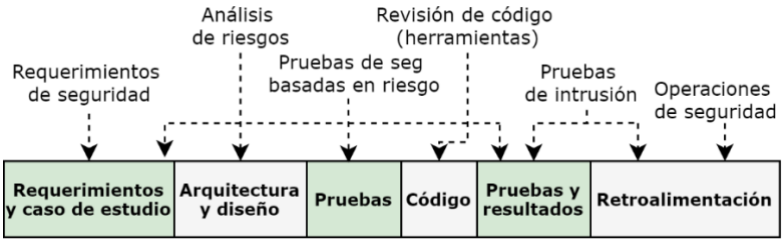



**Figura 1-16.** Aplicación de seguridad en puntos estratégicos del ciclo de vida del desarrollo de software. Adaptada de **[17]**

En el artículo “Securing for DevOps deployment processes: Defenses, risks, research directions” [80], se examinan las implicaciones de seguridad de dos de las prácticas claves de DevOps, automatización del pipeline “deployment” usando una cadena de herramientas para el despliegue e infraestructura como código para especificar el ambiente del software desplegado. El enfoque está representado en identificar qué cambios son relevantes cuando se despliegue manualmente, y cuando se despliega automáticamente mediante DevOps. Para esto se proponen tres (3) casos de estudio usando herramientas de DevOps, configuraciones básicas y examinación de los controles de seguridad provenientes por las herramientas de seguridad.

El modelo de cadena de herramientas (toolchain) usado para los tres (3) casos de uso está conformado por los siguientes componentes: servidor de construcción (infraestructura como código, contenido de aplicación), servicio de aprovisionamiento, herramienta de almacenamiento, almacenamiento de artefactos, servicio de despliegue, máquina virtual provisionada (agente de despliegue). Para este objetivo, se hace uso de tres casos de estudio, con diferentes arquitecturas, aproximaciones para el despliegue de la aplicación y análisis de seguridad para cada uno:

1. Toolchain A - CloudFormation, CodeDeploy, S3, Bash.
2. Toolchain B - Chef Server, Chef Knife, Chef Client, Knife EC2 Plugin, Bash.
3. Toolchain C - Docker, CloudFormation, S3, Bash.

Por último, los autores Bass, Holz, Rimba, Tran y Zhu [15] en el evento "RELENG 2014 Q&A" realizaron la pregunta "¿cuál es la mayor preocupación?", obteniendo como respuesta "alguien que pueda subversionar nuestro pipeline CD". A partir de esto, surge la motivación de explorar cuál es el significado de subversionar un pipeline y así mismo proporcionar algunos escenarios diferentes de subversión. Luego se enfocarán en los problemas de seguridad encontrados en el pipeline. Para lograr el objetivo de endurecer la integridad del pipeline, se proponen tres puntos: 1) no se pueden cambiar el orden de las acciones sin credenciales apropiadas. 2) Solo con las credenciales apropiadas se puede cambiar una acción, claro está, que esta acción no conforme una acción descrita en la especificación original. 3) El resultado de la acción es exactamente como se describe en la más reciente especificación. Además, se propone un endurecimiento (hardening) de seguridad sobre el pipeline, el cual debe seguir estos pasos:

1. Identificar requerimientos de seguridad para el pipeline.
2. Identificar los componentes confiables y no confiables del pipeline.
3. Repetir hasta que se hayan cumplido todos los requerimientos o se puedan descomponen los componentes no confiables.
    a. Modelar la interacción entre componentes.
    b. Analizar el modelo y revisar si este satisface los requerimientos.
    c. Descomponer los componentes no confiables que causan un requerimiento insatisfecho en una porción confiable y no confiable.
4. Implementar nuevos componentes confiables y modificar los componentes no confiables para realizar operaciones sensibles.

Como resultado, se proporciona el proceso recomendado, el cual está basado en tener componentes confiables mediante el acceso a porciones sensibles del pipeline desde otros componentes, que pueden ser no confiables. La construcción diseñada para mejorar la seguridad en el pipeline CD involucra herramientas como Chef, Jenkins, Docker, GitHub y AWS, entre otras.

# 2. Metodología

A continuación, se presentan los procedimientos y actividades realizados para el desarrollo de cada uno de los objetivos específicos planeados. El enfoque metodológico llevó a cabo en forma de análisis, interpretación y presentación de los resultados obtenidos bajo cada prueba desarrollada.

Para el desarrollo del proyecto se consideran las siguientes cuatro (4) fases, como se observa en la Figura 2-1.

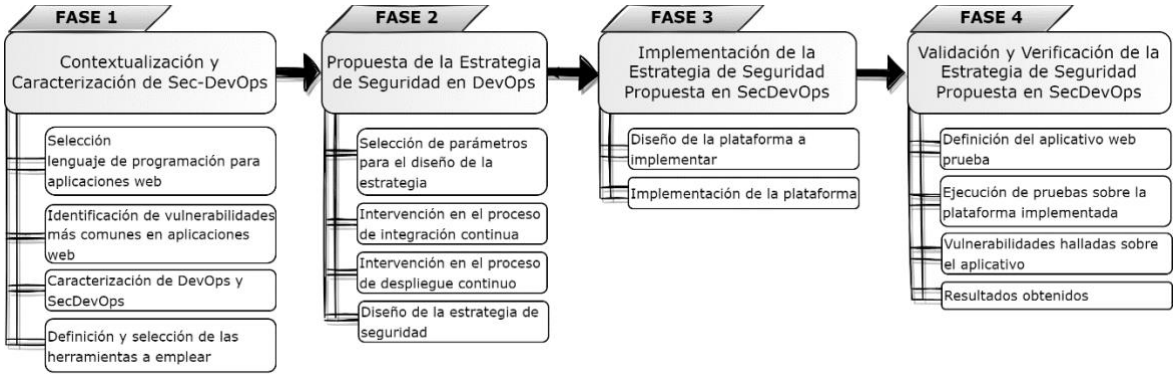



**Figura 2-1.** Fases para el desarrollo del proyecto de grado

## 2.1. Fase 1. Contextualización y caracterización de Sec-DevOps

En esta fase se desarrollan las tareas específicas necesarias para contextualizar y caracterizar los conceptos de DevOps y SecDevOps a lo largo del documento. Este desarrollo está basado en el estudio del estado del arte y las fuentes bibliográficas que aportaron a un mejor entendimiento de estas. Adicionalmente, se plantea el procedimiento para la selección del lenguaje de programación para aplicaciones web.

Esta fase apunta a dar cumplimento al objetivo uno (1), seleccionando el lenguaje de programación, y parte del objetivo dos (2), con la contextualización y caracterización de SecDevOps.

### 2.1.1. Selección del lenguaje de programación para aplicaciones web

En esta actividad se realiza una búsqueda en páginas web enfocadas al desarrollo y en comunidades de desarrolladores, donde se realizan anualmente encuestas para hacer un sondeo de la tecnología y el lenguaje de aplicaciones web más usados, basados en la información de una de las comunidades de desarrolladores más comunes a nivel mundial: *Stack Overflow. "*Stack Overflow es una comunidad abierta

para cualquier persona que desarrolle. Donde se ayuda a obtener respuestas a preguntas relacionadas al desarrollo, compartir conocimiento con los diferentes desarrolladores y así enfocar a encontrar el próximo trabajo de ensueños" [81]. La encuesta anual a desarrolladores de Stack Overflow es la encuesta más grande y completa de personas que desarrollan a nivel mundial. Abarca varios temas, desde las tecnologías favoritas por los desarrolladores, hasta por sus preferencias laborales.

Adicionalmente, se hizo uso de otras fuentes con buena reputación en el mercado del desarrollo de software, encargadas de clasificar y categorizar la popularidad de los lenguajes de programación. Para este caso se seleccionaron tres (3) fuentes adicionales de las mejores listadas en el mercado, cada una con su propia metodología para la clasificación. De las fuentes seleccionadas se incluyen desarrolladores de la firma RedMonk, la cual basa su metodología en la revisión de los repositorios de código de GitHub y en las discusiones de *Stack Overflow*; los analistas de desarrollo de la firma SlashData y, por último, la comunidad de programación TIOBE, la cual se actualiza mensualmente y está basada en la cantidad de ingenieros calificados, cursos y proveedores identificados alrededor del mundo, a través de consultas de motores de bases de búsquedas [82].

En el apartado de resultados *Selección del lenguaje de programación*, capítulo 3.1.1 se encuentra el análisis de dichas encuestas y el lenguaje seleccionado.

### 2.1.2. Identificación de las vulnerabilidades más comunes en aplicaciones web

A partir de la revisión de la bibliografía, se concluye que OWASP es el estándar referente para la clasificación de las vulnerabilidades más comunes en aplicaciones. Dichas vulnerabilidades se encuentran en el apartado del marco teórico *Seguridad en aplicaciones web- Vulnerabilidades en aplicaciones web*, ubicado en el capítulo 1.1.4.

### 2.1.3. Herramientas DevOps y SecDevOps con la caracterización del uso de cada una

Basados en el análisis del estado del arte, se construyó la Tabla 3-3. En la cual se consolidan las herramientas utilizadas en DevOps y SecDevOps. Además, dicha tabla contiene la descripción de las herramientas, componente de seguridad al que aplica, clasificación en DevOps, SecDevOps o ambas y clasificación de acuerdo con los procesos de CI o CD. Se encuentra en el apartado de resultados, *Caracterización de las herramientas DevOps – SecDevOps*, capítulo 3.1.2.

### 2.1.4. Selección de herramientas para la plataforma DevOps y SecDevOps

A partir de la Tabla 3-3, se encuentran las herramientas categorizadas de acuerdo con los componentes de la plataformas DevOps y SecDevOps: orquestador y/o integrador, repositorio centralizado, gestor de paquetes o compilación, repositorio para el almacenamiento de artefactos, calidad de código, pruebas unitarias, configuración y virtualización de entornos. Adicionalmente, apoyados en el resultado del capítulo 3.1.1, donde se define el lenguaje de programación Java el cual es el punto de partida para la selección del orquestador y/o integrador, se sugieren los criterios: soporte de código abierto "open source", versiones pagas, alcance para los procesos CI y CD, popularidad en el mercado, facilidad de uso y buena documentación. A partir de esto, se toman los diferentes orquestadores suministrados por el estado del arte y la bibliografía y se elige el que cumple con la mayoría de los criterios. En base al orquestador y/o integrador seleccionado, la selección de las demás herramientas deben ser integrables con este y deben ofrecer la disponibilidad de los diferentes plugines.

La evaluación con respecto a los criterios mencionados y la selección de cada una de las herramientas que componen la plataforma DevOps y SecDevOps, se presentan en el apartado de resultados, *Selección de herramientas para la plataforma DevOps y SecDevOps*, ubicado en el capítulo 3.1.3.

## 2.2. Fase 2. Propuesta de la estrategia de seguridad en DevOps

Basados en los estudios realizados en implementaciones de la metodología SecDevOps reportados en las fuentes bibliográficas, se logró evidenciar que las estrategias de seguridad propuestas para llevar acabo dicha metodología dependen profundamente de la madurez y capacidad del equipo de seguridad. Apoyado en esto, a continuación, se describen las actividades a desarrollar para la construcción de la estrategia de seguridad propuesta en DevOps.

**Observación:** Como en la intervención llevada acabo en el ciclo DevOps a lo largo de esta investigación, se han incorporado diferentes elementos como son; criterios, proceso, fases, flujos, ciclos, componentes y herramientas, por ende, para integrar todos estos conceptos en la estrategia de seguridad propuesta se optó por agruparlos bajo el término parámetros, dado que, en la definición dada por la Real Academia de la Lengua Española es: "Dato o factor que se toma como necesario para analizar o valorar una situación" [83]. Por consiguiente, el concepto de parámetros puede recoger los diferentes criterios antes expuestos.

### 2.2.1. Selección de parámetros para el diseño de la estrategia e intervención en el proceso de integración continua y despliegue continuo

La referencia básica de la implementación de seguridad en todo el ciclo DevOps dada por la literatura, fue tomada como punto de partida para la construcción de una estrategia de seguridad integral.

La implementación referenciada por la literatura modela en la mayoría de los casos un esquema de seguridad simple a lo largo de todo el ciclo del desarrollo de software. En algunas referencias bibliográficas, sólo se menciona la ejecución de controles de seguridad en ciertos puntos de todo el ciclo, como se presenta a continuación, esto haciendo referencia al tema técnico, pero adicionalmente, no se trabajan temas referentes a como adoptar, mantener y mejorar SecDevOps como cultura.

La estrategia de seguridad propuesta está pensada para tener un cubrimiento desde el punto de vista técnico hasta el tema cultural y, así mismo, que sea adaptable al nivel de madurez de la compañía donde se desee implementar. Apoyados en esto, se exponen los controles de seguridad en la metodología SecDevOps basada en la revisión del estado del arte, los cuales se desglosan para mayor entendimiento en el proceso de CI y el proceso de CD. Para el proceso de integración continua (CI), ver Figura 2-2.

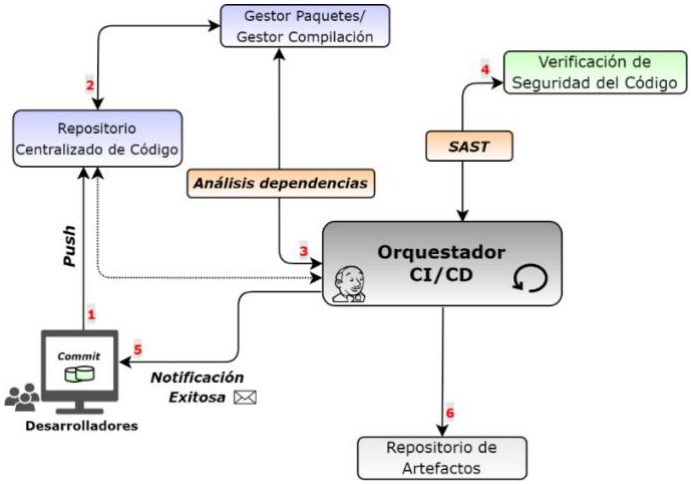



**Figura 2-2.** Estrategias de seguridad en el proceso CI - SecDevOps

Para el proceso de despliegue continuo (CD), ver Figura 2-3.

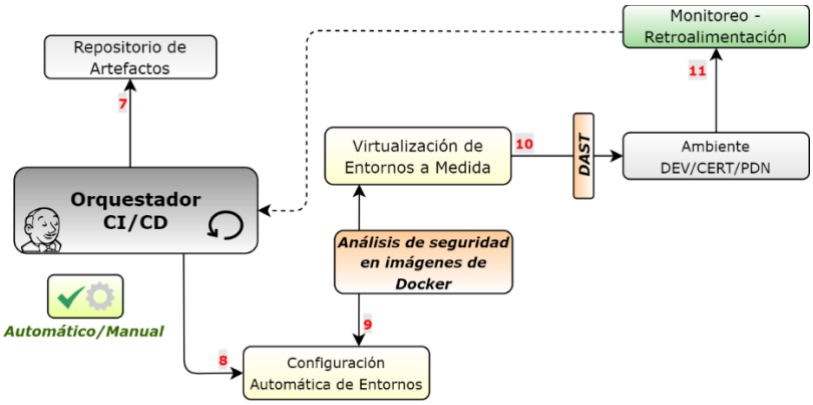



**Figura 2-3.** Estrategias de seguridad en el proceso CD – SecDevOps

A partir de la definición de SecDevOps dada por la literatura, se hace fundamental implementar seguridad desde las etapas tempraneras del ciclo de desarrollo de software hasta su despliegue y puesta en producción, por lo que se proponen nuevos controles de seguridad sumados a los expuestos en la Figura 2-2 y Figura 2-3, que se incorporen tanto en el CI como en el CD, para así lograr un cubrimiento a lo largo de todo el ciclo. Por consecuencia, al incluir nuevos controles sobre el ciclo completo, en seguida se van a generar más hallazgos y vulnerabilidades creando la necesidad de ser gestionadas y administradas por el equipo de seguridad. Además a esto, es esencial disponer de una acción la cual pueda determinar sí de acuerdo al resultado y al apetito de riesgo donde se implemente la estrategia se continúe o se detenga el flujo llevado por DevOps. En este punto, dada la experiencia y el manejo de las diferentes herramientas trabajadas en DevOps y SecDevOps, es de gran importancia disponer de un flujo estándar para ser ejecutado mediante pipelines, el cual pueda contribuir en la correcta implementación de la metodología SecDevOps.

En la práctica no se refleja totalmente la aplicación de la metodología SecDevOps que dé respuesta a las necesidades actuales de la construcción de software, desde los criterios de seguridad informática, debido a que la implementación y la adopción de la cultura dependen de factores como la madurez del equipo, presupuesto destinado para temas de seguridad y carencia de habilidades técnicas de seguridad, por esto, no pueden ser tenidos en cuenta en todo el ciclo DevOps. Teniendo en cuenta lo anterior, se proponen siete (7) parámetros clasificados en: obligatorios y transversales. Los primeros, deben ser implementados en la estrategia de seguridad, y los transversales, pueden adoptarse de acuerdo a los diferentes factores mencionados anteriormente.

Así, luego de lo expuesto, se proponen siete (7) parámetros que componen una estrategia integral de seguridad. Los parámetros de la estrategia de seguridad en DevOps propuesta no sólo están compuestos

de componentes y herramientas sí no que también están basados en controles de seguridad adicionales, políticas, definiciones y criterios. Estos parámetros se detallan en el apartado de resultados *Fase 2. Propuesta de la estrategia de seguridad en DevOps,* ubicado en el capítulo 3.2.1.

### 2.2.2. Diseño de la estrategia de seguridad en DevOps

Con base en la selección de los parámetros que componen la estrategia de seguridad en DevOps, se desarrollan y se detallan cada uno de estos para lograr el diseño total de la estrategia propuesta. El diseño logrado se expone en el apartado de resultados *Diseño de la estrategia de seguridad propuesta*, capítulo 3.2.2.

## 2.3. Fase 3. Implementación de la estrategia de seguridad propuesta en SecDevOps

En esta fase se desarrollan las tareas específicas para el diseño, implementación y ejecución de las plataformas de DevOps y SecDevOps.

### 2.3.1. Diseño de la plataforma de DevOps y SecDevOps

La arquitectura y el diseño propuestos de cada una de las plataformas están basados en el despliegue de los componentes de infraestructura y servicios administrados de Azure, de consumo *open source*.

La plataforma DevOps está fundamentada en las herramientas de software libre seleccionadas en la Tabla 3-11. El componente principal de la plataforma DevOps es el orquestador CI/CD Jenkins, el cual cumple con la función de automatizar los procesos de integración continua y despliegue continuo a partir del despliegue de cada una de las herramientas seleccionadas. Tomando en cuenta esto, se van integrando las diferentes herramientas que componen toda la plataforma. Se inició con la configuración de GitLab, utilizada como repositorio de código centralizado brindando el control de versiones del código y ofreciendo la trazabilidad de los cambios realizados por los desarrolladores. Fundamentados en la selección del lenguaje de programación Java, la herramienta Gradle se desplegó para la compilación y la construcción del código del aplicativo y, paralelamente, el framework Junit es usado para la automatización de pruebas unitarias. Después de ejecutar la construcción del código se generó el archivo ejecutable (.jar) el cual se almacena localmente en la máquina virtual donde se ejecuta Jenkins, cumpliendo la función de repositorio de artefactos local. Luego se despliega la herramienta SonarQube

para el análisis de mantenibilidad y calidad del código. Se crea la imagen Docker basada en el Dockerfile contenido dentro de los recursos del repositorio del código del aplicativo y esta es almacenada en el repositorio de imágenes Docker de Azure (ACR) y, por último, se preparó el ambiente de kubernetes en Azure para proceder a su despliegue. El resultado del diseño de la plataforma DevOps se presenta apartado de resultados *Diseño de la plataforma de DevOps y SecDevOps* ubicado en el capítulo 3.3.1.

Por otro lado, la plataforma SecDevOps está basada en la plataforma DevOps y las herramientas de software libre seleccionadas en: Tabla 3-16, Tabla 3-17, Tabla 3-21, Tabla 3-22. Para la construcción de esta se realiza un desglose de las herramientas de seguridad seleccionadas describiendo su función específica tanto para el CI como para el CD y así lograr el diseño definitivo, expuesto en el apartado de resultados *Diseño de la plataforma de DevOps y SecDevOps* ubicado en el capítulo 3.3.1.

### 2.3.2. Implementación de las plataformas DevOps y SecDevOps

Partiendo de las ventajas ofrecidas en cuanto a la flexibilidad, accesibilidad y facilidad en el despliegue de infraestructura por parte de las tecnologías nube (cloud), se describe a continuación el paso a paso empleado para la implementación de las plataformas de DevOps y SecDevOps propuestas en la metodología.

Desde el diseño conceptual expuesto en la *Fase 2. Propuesta de la estrategia de seguridad en DevOps*, capítulo 2.2, se inicia con el despliegue de cada una de las herramientas seleccionadas en la Tabla 3-11 que componen la plataforma DevOps. Posteriormente, se procede a realizar la integración de estas con el orquestador Jenkins para así comprobar su funcionamiento. De igual manera, este procedimiento es empleado para el despliegue de la plataforma SecDevOps.

La implementación, configuración e integración de cada una de las plataformas de DevOps y SecDevOps se presentan en el apartado de resultados, *Implementación de las plataformas DevOps y SecDevOps* ubicado en el capítulo 3.3.2.

## 2.4. Fase 4. Validación y verificación de la estrategia de seguridad propuesta en SecDevOps

En esta fase se define el aplicativo web usado en las pruebas propuestas para la validación y verificación de la estrategia de seguridad expuesta en el capítulo 3.2. Adicionalmente, esta fase apunta a dar

cumplimento total al objetivo tres (3), llevando a cabo la selección del aplicativo web y la ejecución de las respectivas pruebas sobre las plataformas desplegadas.

### 2.4.1. Definición del aplicativo web prueba

Se procede a la selección del aplicativo web para ser construido, ejecutado y desplegado a lo largo de las plataformas DevOps y SecDevOps configuradas. Para la selección de este, se eligen los siguientes requerimientos: el aplicativo web debe estar desarrollado bajo el lenguaje de programación Java, contar con algún modelo de arquitectura de software básico, uso de un framework de licencia libre para su construcción, incluir ciertas buenas prácticas de desarrollo, disponer de un modelo de datos para su funcionamiento y, por último, que el aplicativo pueda ser desplegado y certificado en ambientes pre-productivos y, posteriormente, lograr puesta en producción. Basados en esto, la descripción del aplicativo web se expone en el apartado de resultados *Definición aplicativo web prueba*, ubicado en el capítulo 3.4.1.

### 2.4.2. Ejecución de pruebas para validar la estrategia de seguridad propuesta

Para la validación y verificación de la estrategia de seguridad propuesta SecDevOps se plantean las siguientes tres (3) pruebas que serán ejecutadas bajo las plataformas de DevOps y SecDevOps configuradas. La validación se realiza al aplicativo web o producto que se desplegaría en producción obtenido al final del proceso CI y CD de DevOps o SecDevOps. Analizando la cantidad de hallazgos o posibles vulnerabilidades que posee el aplicativo en cada una de las tres pruebas. Al comparar los tres aplicativos obtenidos en cada uno de las tres pruebas se puede validar y verificar la metodología propuesta.

- ***Prueba 1. Aplicativo web ejecutado y desplegado sobre la plataforma DevOps***

El software seleccionado es construido, ejecutado y desplegado empleando la plataforma DevOps configurada. Sin implementar ningún control de seguridad.

- ***Prueba 2. Pruebas de seguridad sobre el aplicativo web desplegado sobre la plataforma DevOps***

Inicialmente se procede a la definición e instalación de las herramientas para ejecutar el análisis estático (SAST) y *pentesting* sobre el aplicativo desplegado en la prueba 1. La instalación de estas y la ejecución de ambos análisis se exponen en el apartado de resultados, ubicado en el capítulo 3.4.2. Las herramientas de

seguridad seleccionadas para ejecutar esta prueba son diferentes a las elegidas para la plataforma SecDevOps.

- ***Prueba 3. Aplicativo web ejecutado y desplegado sobre la plataforma SecDevOps***

Partiendo de la estrategia de seguridad construida y expuesta conceptualmente en la *Fase 2. Propuesta de la estrategia de seguridad en DevOps*, capítulo 2.2, se procede a construir y desplegar el software a través de la plataforma SecDevOps preparada.

El procedimiento llevado acabo para cumplir con cada una de las pruebas propuestas se expone en el apartado de resultados *Ejecución de pruebas sobre las plataformas implementadas*, capítulo 3.4.2.

### 2.4.3. Hallazgos encontrados y resultados obtenidos

Los hallazgos y resultados obtenidos de cada una de las pruebas mencionadas son consolidados y expuestos en el apartado de resultados *Hallazgos encontrados y resultados obtenidos,* ubicado en el capítulo 3.4.3.

# 3. Resultados

## 3.1. Fase 1. Contextualización y caracterización de SecDevOps

A continuación, se presentan los resultados obtenidos al aplicar el procedimiento expuesto en la metodología.

### 3.1.1. Selección del lenguaje de programación

A partir de las encuestas desarrolladas desde el año 2013, se realizó un análisis de los lenguajes más comunes y votados en cada uno de ellos.

- ***Análisis de las encuestas del año 2013 al 2020 realizadas por la comunidad Stack Overflow***

*Resultados de las encuestas de los años 2013 – 2014 – 2015*

Para el año 2013 se obtuvieron 8042 respuestas; para el año 2014 se obtuvieron 6537 respuestas y, para el año 2015, 26086 respuestas fueron registradas. Participaron personas de aproximadamente 157 países en la encuesta de 45 preguntas, donde 6800 personas fueron identificadas como desarrolladores *fullstack,* 1900 como desarrolladores móviles, 1200 como desarrolladores *front-end* y 12000 clasificados categoría general; para visualizar el porcentaje de cada tecnología calificada por cada desarrollador, se expone la Figura 3-1.

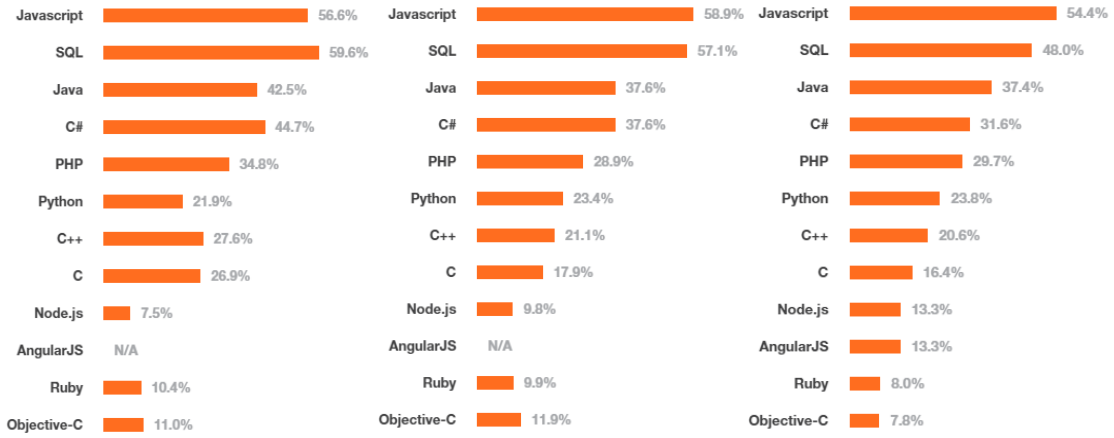



**Figura 3-1.** Resultados de la votación de las tecnologías más importantes para los años 2013/2014/2015 **[83]**

De acuerdo con los resultados obtenidos en las encuestas de los años 2013, 2014 y 2015, el lenguaje JavaScript se mantiene como número uno consecutivamente. A los lenguajes NodeJS y AngularJs se les observa un comportamiento de desaparición progresiva y el lenguaje Java se mantiene en el tercer lugar y como una de las etiquetas más buscadas en *Stack Overflow* [83].

*Resultados de la encuesta del año 2016*

En este año se cuentan con más de 50000 desarrolladores que aportaron a la encuesta y compartieron experiencias relacionadas al desarrollo. A continuación, se comparten los resultados de 6474 respuestas en total, que se pueden observar en la Figura 3-2.

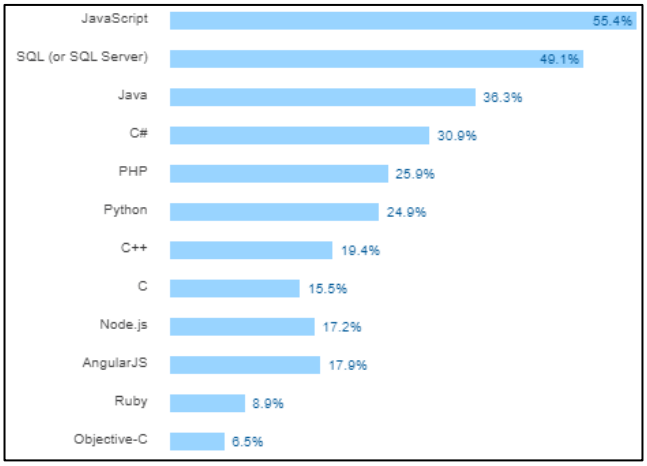



**Figura 3-2.** Tecnologías más populares en la encuesta realizada en el 2016 **[84]**

Además de que se conserva un comportamiento similar al año anterior, en este año se adicionan dos categorías nuevas en las encuestas realizadas: tecnología superior y tendencias tecnológicas en *Stack Overflow*. Los lenguajes de programación JavaScript y Java corresponden a la primera nueva tecnología y continúan siendo los dos lenguajes más votados por los desarrolladores. En cuanto a la segunda tecnología adicionada, surgen dos lenguajes de programación que no fueron mencionados en los años anteriores: React y Spark [84].

*Resultados de la encuesta del año 2017*

Para este año más de 64000 desarrolladores compartieron sus enseñanzas, experiencias del uso de herramientas y opiniones de nuevas herramientas identificadas, bajo esto se registraron 36625 respuestas, las cuales se observan en la Figura 3-3.

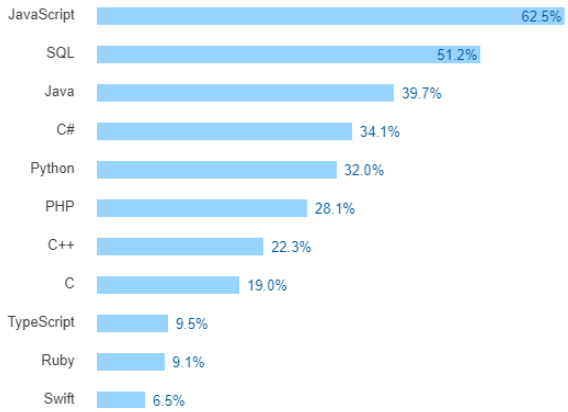



**Figura 3-3.** Tecnologías más populares en la encuesta realizada en el 2017 **[85]**

Por cuarto año consecutivo se observa la preferencia por el uso del lenguaje JavaScript, igual que los dos siguientes lenguajes que le siguen: SQL y Java. Cabe notar que hay un crecimiento en el uso de Python

superando a PHP. En este año se recopilaron los resultados obtenidos desde el 2013 hasta el 2017, dando como resultado una visualización del comportamiento de los diferentes lenguajes de programación. En la Figura 3-4 se observa cómo ha crecido la popularidad de los lenguajes como JavaScript, Python, y Node.JS; por el contrario, Java se mantiene estable en estos cinco años.

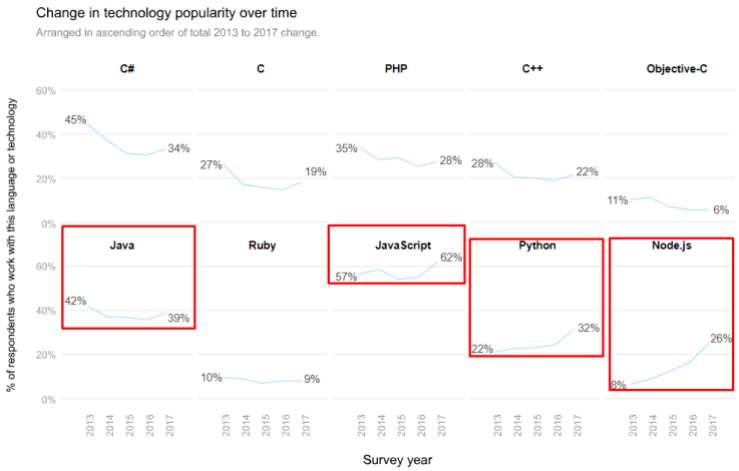



**Figura 3-4.** Comportamiento de los lenguajes de programación entre los años 2013-2017 **[85]**

*Resultados de la encuesta del año 2018*

En este año, más de 100000 desarrolladores participaron en la encuesta realizada, contando las vivencias del aprendizaje con los lenguajes de programación, las estrategias utilizadas para adquirir experiencia y las herramientas usadas para mejorar las habilidades de desarrollo. Cabe mencionar que a partir de este año empezaron a ser tendencia en la industria términos como: DevOps, Machine Learning, los lenguajes y frameworks asociados con estas tendencias; además, el nivel salarial obtenido en estas es mucho más alto con respecto de las tendencias convencionales. Para este año se registraron 78334 respuestas, las cuales se pueden ver reflejadas en la Figura 3-5.

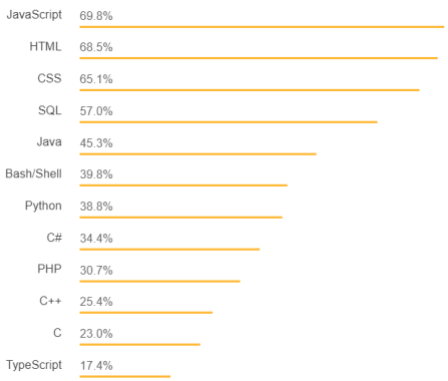



**Figura 3-5.** Tecnologías más populares en la encuesta realizada en el 2018 **[86]**

Por sexto año consecutivo, el lenguaje de programación JavaScript es el lenguaje más popular dentro de los votantes en las encuestas realizadas, otra novedad importante es que Python ha superado tanto como a PHP y a C#, lo cual se concluye que tiene un gran crecimiento y apogeo dentro de los desarrolladores [86].

*Resultados de la encuesta del año 2019*

Para este año, cerca de 90000 desarrolladores contestaron la encesta. Se registraron 87357 respuestas. Los resultados se pueden observar en la Figura 3-6.

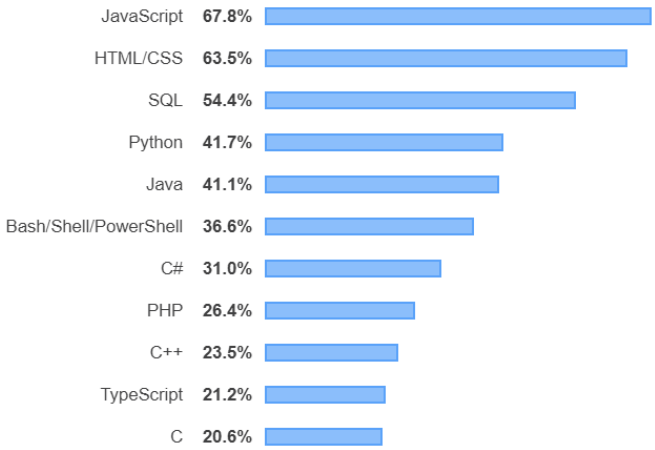



**Figura 3-6.** Tecnologías más populares en la encuesta realizada en el 2019 **[87]**

Continua JavaScript como el lenguaje más común, y Python sigue con el crecimiento año a año que lo ubica en el puesto cuatro (4), y sobrepasando a lenguajes como Java.

*Resultados de la encuesta del año 2020*

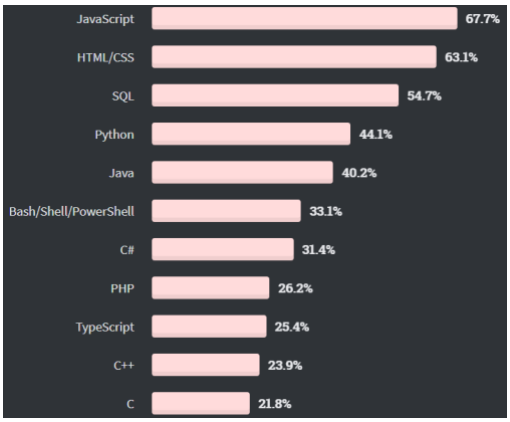



**Figura 3-7.** Tecnologías más populares en la encuesta realizada en el 2020 **[88]**

Continua JavaScript como el lenguaje más común y Python sigue con el crecimiento año a año que lo ubica en el cuarto puesto superando a lenguajes como Java

Dada la tendencia de los años anteriores, el lenguaje de programación JavaScript ha mantenido la fuerza como el más popular y usado por los desarrolladores que respondiendo las encuestas [88].

Adicionalmente, en el mercado existen distintas fuentes con buena reputación encargadas en clasificar y categorizar la popularidad de los lenguajes de programación. Para este caso se seleccionaron cuatro de las mejores fuentes listadas en el mercado, cada una con su propia metodología empleada para la clasificación de los lenguajes de programación. Se incluyen desarrolladores de la firma RedMonk, la cual basa su metodología en la revisión de los repositorios de código de GitHub y en las discusiones de *Stack Overflow*, la comunidad de *Stack Overflow*, los analistas de desarrollo de la firma SlashData y, por último, la comunidad de programación TIOBE, la cual se actualiza mensualmente y está basada en la cantidad de ingenieros calificados, cursos y proveedores identificados alrededor del mundo a través de consultas de motores de bases de búsquedas [82]. En la Figura 3-8 se puede visualizar la relación del top 10 del año 2019 de los lenguajes de programación.

| RedMonk | Stack Overflow | SlashData | TIOBE Index 7/19 |
|---|---|---|---|
| JavaScript | JavaScript | JavaScript | Java |
| Java | HTML/CSS | Python | C |
| Python | SQL | Java | Python |
| PHP | Python | C# | C++ |
| Tie: C++/C# | Java | C/C++ | C# |
| | Bash/Shell/Powershell | PHP | Visual Basic.NET |
| CSS | C# | Visual tools | JavaScript |
| Ruby | PHP | Swift | PHP |
| C | TypeScript | Ruby | SQL |
| TypeScript | C++ | Kotlin | Objective C |

**Figura 3-8.** Top 10: 2019. Lenguajes de programación de las cuatro fuentes **[82]**

Siguiendo el enfoque de la comunidad TIOBE, nos apoyamos en el histórico de los índices sobre lenguajes de programación calculados a lo largo de 18 años, desde el año 2002 hasta el año 2020. El comportamiento más estable durante este período de tiempo se debe al lenguaje de programación Java, como se puede observar en la siguiente Figura 3-9.

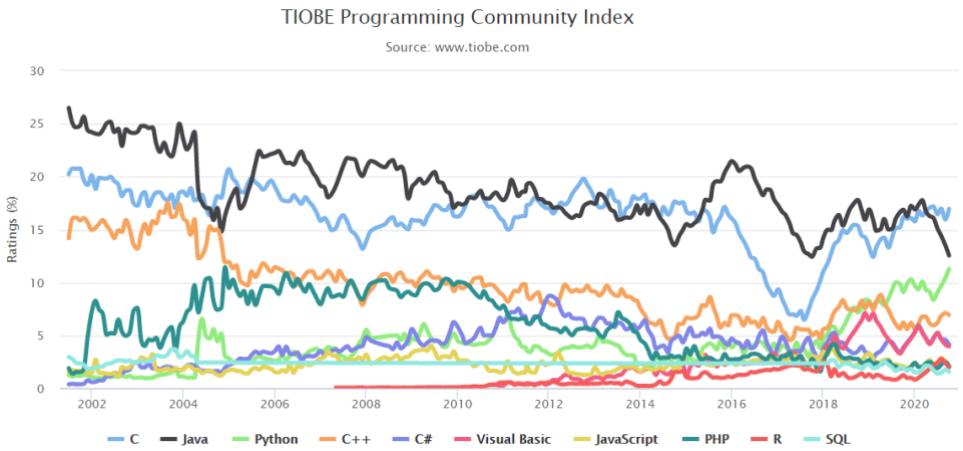



**Figura 3-9.** Histórico de lenguajes de programación de la comunidad TIOBE **[89]**

Además, para confirmar la información suministrada por TIOBE, se lista a continuación el ranking de los 10 lenguajes de programación más populares clasificados a junio de 2020 por esta comunidad (ver Figura 3-10). Es importante aclarar que este ranking está basado en motores de búsqueda populares como: Google, Bing, Yahoo!, Wikipedia, Amazon, Youtube y Baidau, y que la definición para la construcción de este no está referenciada por la premisa de mejor lenguaje de programación o lenguaje con el que se han escrito la mayoría de las líneas de código. El índice TIOBE puede servir de gran ayuda para verificar si las habilidades de programación se encuentran actualizadas y así tomar una decisión estratégica de que lenguaje de programación e debe adoptar al construir y desarrollar un nuevo software [89].

| Jun 2020 | Jun 2019 | Change | Programming Language | Ratings | Change |
|---|---|---|---|---|---|
| 1 | 2 | ^ | C | 17.19% | +3.89% |
| 2 | 1 | v | Java | 16.10% | +1.10% |
| 3 | 3 | | Python | 8.36% | -0.16% |
| 4 | 4 | | C++ | 5.95% | -1.43% |
| 5 | 6 | ^ | C# | 4.73% | +0.24% |
| 6 | 5 | v | Visual Basic | 4.69% | +0.07% |
| 7 | 7 | | JavaScript | 2.27% | -0.44% |
| 8 | 8 | | PHP | 2.26% | -0.30% |
| 9 | 22 | ^^ | R | 2.19% | +1.27% |
| 10 | 9 | v | SQL | 1.73% | -0.50% |

**Figura 3-10.** Top 10 índice TIOBE a junio de 2020 **[89]**

En función del estudio se revisaron las comunidades más populares y con mayor soporte en el ámbito del desarrollo de software. Luego de evidenciar el comportamiento de los lenguajes de programación, basados en las encuestas desarrolladas por la comunidad más grande de desarrollo del mundo, *Stack Overflow*, durante ocho (8) años consecutivos (2013-2020), se tomaron en cuenta los ocho lenguajes de programación que se mantuvieron constantes durante este período, siendo: JavaScript, SQL, C#, Java, PHP, C++, C y Python. Posteriormente a la selección de los ocho lenguajes de programación, se tomó el

porcentaje de popularidad de cada uno por cada año, se promedió y se organizó de mayor a menor, como se visualiza en la Tabla 3-1.

**Tabla 3-1.** Clasificación de los ocho lenguajes de programación más constantes

| Nro. | Lenguajes seleccionados | 2013 | 2014 | 2015 | 2016 | 2017 | 2018 | 2019 | 2020 | Promedio |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | JavaScript | 56,6 | 58,9 | 54,4 | 55,4 | 62,5 | 69,8 | 67,8 | 67,7 | **61,6** |
| 2 | SQL | 59,6 | 57,1 | 48,0 | 49,1 | 51,2 | 57,0 | 54,4 | 54,7 | **53,9** |
| 3 | Java | 42,5 | 37,6 | 37,4 | 36,3 | 39,7 | 45,3 | 41,1 | 40,2 | **40,0** |
| 4 | C# | 44,7 | 37,6 | 31,6 | 30,9 | 34,1 | 34,4 | 31,0 | 31,4 | **34,5** |
| 5 | Python | 21,9 | 23,4 | 23,8 | 24,9 | 32,0 | 38,8 | 41,7 | 44,1 | **31,3** |
| 6 | PHP | 34,8 | 28,9 | 29,7 | 25,9 | 28,1 | 30,7 | 26,4 | 26,4 | **28,9** |
| 7 | C++ | 27,6 | 21,1 | 20,6 | 19,4 | 22,3 | 25,4 | 23,5 | 23,9 | **23,0** |
| 8 | C | 26,9 | 17,9 | 16,4 | 15,5 | 19,0 | 23,0 | 20,6 | 21,8 | **20,1** |

De la Tabla 3-1, es correcto aclarar el enfoque a partir del cual se orientó esta investigación es aplicaciones web, por ende, se eliminó el lenguaje de programación SQL, ya que este es usado para el entorno de bases de datos. Siguiendo esto, la Tabla 3-1 se modificó y los lenguajes de programación fueron ordenados de mayor a menor, de acuerdo con el promedio, como se observa en la
Tabla 3-2.

**Tabla 3-2.** Consolidación de lenguajes de programación orientados a aplicaciones web

| Nro. | Lenguajes seleccionados | Promedio |
|---|---|---|
| 1 | JavaScript | **61,6** |
| 2 | Java | **40,0** |
| 3 | C# | **34,5** |
| 4 | Python | **31,3** |
| 5 | PHP | **28,9** |
| 6 | C++ | **23,0** |
| 7 | C | **20,1** |

Tomando como base la Figura 3-9, donde se evidenció que el lenguaje de programación más estable durante un período de tiempo de 12 años, según TIOBE es Java y, de la

Tabla 3-2 donde se consolidaron las encuestas realizadas por la comunidad *Stack Overflow*, se concluye que ***El lenguaje de programación seleccionado para el cumplimiento de la fase 1 – actividad 1 es: Java.***

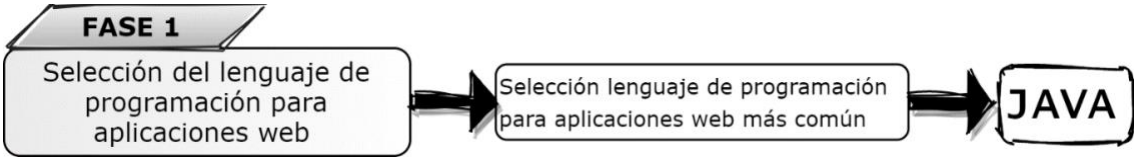



**Figura 3-11.** Resultado Fase 1 – Actividad 1

### 3.1.2. Caracterización de las herramientas DevOps – SecDevOps

A continuación, en la Tabla 3-3 se presentan las herramientas consolidadas a partir de la revisión del estado del arte, especificando la clasificación, descripción y el componente y/o estrategia de seguridad en el que interviene en el ciclo DevOps o SecDevOps respectivamente.

**Tabla 3-3.** Herramientas DevOps – SecDevOps con su caracterización

| Herramientas | DevOps | Seg. en DevOps | Integración Continua | Despliegue Continuo | Descripción de la herramienta | Componente y/o Estrategia de seguridad |
|---|---|---|---|---|---|---|
| *Checkmark* | | X | X | X | Checkmarx, ofrece la plataforma de seguridad de software más completa del sector que se unifica con DevOps y proporciona pruebas de seguridad de aplicaciones estáticas e interactivas, análisis de composición de software y programas de formación y concienciación de AppSec para desarrolladores con el fin de reducir y remediar el riesgo de las vulnerabilidades de software [19]. | CxSAST, CxSCA, CxIAST |
| *Veracode* | | X | X | X | Veracode, ofrece un camino escalable y holístico para manejar los riesgos de seguridad a través de un portafolio de aplicación entero [19]. | SAST - SCA - DAST |
| *ShiftLeft (NextGen Static Analysis Secures Every Pull Request)* | | X | X | | Liberación rápida con seguridad - NG-SAST [19]. | SAST |
| *Jenkins* | X | | X | X | Servidor de automatización de código abierto “open source” líder. Jenkins provee cientos de plugines para soportar la construcción, despliegue y automatización de cualquier proyecto [19]. | Integrador CI/CD |
| *Docker* | X | | | | Un contenedor es una unidad de software estándar que empaqueta el código y todas sus dependencias para que la aplicación se ejecute de forma rápida y fiable de un entorno informático a otro. Una imagen de contenedor Docker es un paquete de software ligero, independiente y ejecutable que incluye todo lo necesario para ejecutar una aplicación: código, tiempo de ejecución, herramientas del sistema, bibliotecas del sistema y configuraciones [19]. | Virtualización de Entornos a Medida |
| *Splunk* | X | | | | Splunk, es una plataforma que permite la investigación, monitoreo, análisis y acción [19]. | Monitoreo - Retroalimentación |

| | | | | | | |
|---|---|---|---|---|---|---|
| *SonarQube* | X | X | X | | SonarQube, fortalece a todos los desarrolladores a escribir código más limpio y seguro. Además, cuenta con una comunidad de más de 200k de desarrolladores alrededor del mundo [19]. | Verificación de Calidad del Código - SAST |
| *OWASP Zed Attack Proxy (ZAP)* | | X | | X | Es la aplicación para escanear web más usado alrededor del mundo [19]. | DAST |
| *Clair* | | X | | X | Clair, es un proyecto open source para el análisis estático de vulnerabilidades en aplicaciones y contenedores Docker [19]. | Análisis imágenes Docker |
| *Klair* | | X | | X | Klair, es una herramienta sencilla para analizar imágenes almacenadas en un registro Docker privado o público en busca de vulnerabilidades de seguridad utilizando Clair. Klair, está diseñado para ser utilizado como una herramienta de integración por lo que se basa en las variables de entorno. Es un único binario que no requiere dependencias [19]. | Análisis imágenes Docker |
| *OWASP Dependency Check* | | X | X | | Dependency-Check, es una herramienta de análisis de composición de software que intenta detectar las vulnerabilidades divulgadas contenidas en las dependencias de los proyectos [19]. | Análisis de dependencias |
| *Warnings Next Generation* | | X | X | | El plugin Jenkins Warnings Next Generation recoge las advertencias del compilador o los problemas reportados por las herramientas de análisis estático y visualiza los resultados [16]. | SAST |
| *Apache Maven* | X | | | | Apache Maven, es una herramienta de gestión y comprensión de proyectos de software. Basado en el concepto de modelo de objetos de proyecto (POM), Maven puede gestionar la construcción, los informes y la documentación de un proyecto desde una pieza central de información [18]. | Gestor Paquetes/ Gestor Compilación |
| *OWAS ZAP Spider* | | X | | X | La 'araña' es un herramienta usada para descubrir automáticamente nuevos recursos (URL) en un sitio en particular [16]. | DAST |
| *HTML Publisher* | X | | | | El complemento HTML Publisher es usado para publicar reportes HTML que la construcción genera durante el trabajo de construcción de páginas. Este es designado para trabajar con proyectos de estilo libre, y además puede ser usado en el pipeline de Jenkins [16]. | Plugin Integrador CI/CD |
| *Anchore Container Image Scanner* | | X | | X | Anchore, es una plataforma para la inspección y analítica de contenedores que habilita operadores para analizar, inspeccionar, realizar escaneos de seguridad, y evaluar políticas personalizadas contra imágenes de contenedores [16]. | Análisis imágenes Docker |
| *Chef* | X | | | | Chef automatiza la configuración de infraestructura, asegurando que cada sistema es configurado correctamente [15]. | Configuración Automática de Entornos |
| *Github* | X | | | | GitHub, es una plataforma de desarrolladores inspirada para trabajar en colaboración [15]. | Repositorio Centralizado de Código |

| | | | | | | |
|---|---|---|---|---|---|---|
| *AWS* | X | | | | Ofrece confianza, escalabilidad, y servicios de computo en nube no costosos [15]. | Virtualización de Entornos a Medida |
| *Kubernetes* | X | | | | Plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios [22]. | Virtualización de Entornos a Medida |
| *Teamcity* | X | | | | TeamCity, es un servidor de gestión de compilación e integración continua de JetBrains [22]. | Integrador CI/CD |
| *Spinnaker* | X | | | | Spinnaker, es una plataforma de entrega continua multi-nube de código abierto para liberar cambios de software con alta velocidad y confianza [22]. | Integrador CI/CD |
| *Travis CI* | X | | | | Travis CI, es un servicio de integración continua alojado que se utiliza para construir y probar proyectos de software alojados en GitHub y Bitbucket [22]. | Integrador CI/CD |
| *Origin Continuous Deployment (OCD)* | X | | | X | Origin Continuous Deployment (OCD), es una herramienta Bash y paquetes Helm para construir, configurar y desplegar repetidamente aplicaciones en Origin Kubernetes Distribution y Red Hat OpenShift impulsado por git webhooks [22]. | Integrador CI/CD |
| *Concourse CI* | X | | | | Es un programa de código abierto para la automatización continua. Construido sobre la simple mecánica de los recursos, las tareas y los trabajos, Concourse presenta un enfoque general de la automatización que lo hace grande para CI/CD [22]. | Integrador CI/CD |
| *JFrog Artifactory* | X | | | | JFrog Artifactory, es el corazón de misión crítica de la plataforma JFrog que funciona como la única fuente de verdad para todos los paquetes, imágenes de contenedores y gráficos Helm, a medida que se mueven a través de toda la tubería DevOps [22]. | Repositorio de Artefactos |
| *PMD* | | X | X | | Un analizador de código estático extensible entre lenguajes [22]. | SAST |
| *Checkstyle* | | X | X | | Checkstyle, es una herramienta de desarrollo que ayuda a los programadores a escribir código Java que se adhiere a un estándar de codificación. Automatiza el proceso de comprobación del código Java para ahorrar a los humanos esta aburrida (pero importante) tarea. Esto lo hace ideal para proyectos que quieren imponer un estándar de codificación [22]. | SAST |
| *FindBugs* | | X | X | | Esta es la página web de FindBugs, un programa que utiliza el análisis estático para buscar errores en el código Java [22]. | SAST |
| *OWASP Find Security Bugs* | | X | X | | El complemento SpotBugs para seguridad audita aplicaciones web desarrolladas en Java [22]. | SAST |
| *BDD Security* | | X | X | | BDD-Security es un marco de pruebas de seguridad que utiliza los conceptos del Desarrollo Dirigido por el Comportamiento para crear especificaciones de seguridad auto verificables [22]. | Framework Seguridad |

| | | | | | | |
|---|---|---|---|---|---|---|
| *JFrog Xray* | | **X** | **X** | **X** | Análisis de seguridad continua en repositorio de artefactos [22]. | Seguridad en Repositorio de Artefactos |
| *Security Monkey* | | **X** | | | Security Monkey supervisa sus cuentas de AWS y GCP en busca de cambios en las políticas y alertas sobre configuraciones inseguras [22]. | Seguridad en políticas (AWS/GCP) |
| *Black Duck* | | **X** | **X** | | Black Duck ofrece una solución integral de análisis de composición de software (SCA) para gestionar la seguridad, la calidad y el riesgo de cumplimiento de licencias que se deriva del uso de código abierto y de terceros en aplicaciones y contenedores [90]. | SCA - Análisis de dependencias |
| *Snyk* | | **X** | | **X** | Snyk, es una plataforma de seguridad para desarrolladores para asegurar código, dependencias, contenedores e infraestructura cómo código [91]. | Seguridad imágenes Docker - Análisis de dependencias |
| *Git SCM* | **X** | | | | Git, es un sistema de control de versiones distribuido, gratuito y de código abierto, diseñado para gestionar desde proyectos pequeños a muy grandes con rapidez y eficacia [18]. | Repositorio Centralizado de Código |
| *BitBucket* | **X** | | | | Bitbucket, es más que una simple gestión de código Git. Bitbucket ofrece a los equipos un lugar para planificar proyectos, colaborar en el código, probar y desplegar [18]. | Repositorio Centralizado de Código |
| *Gradle* | **X** | | | | Gradle, es una herramienta de automatización de construcción de código abierto que está diseñada para ser lo suficientemente flexible como para construir casi cualquier tipo de software. A continuación se presenta un resumen de alto nivel de algunas de sus características más importantes [18]. | Gestor Paquetes/ Gestor Compilación |
| *GOCD* | **X** | | | | Servidor CI/CD open source y libre de licencia [18]. | Servidor CI/CD - Integrador - Orquestador |
| *Puppet* | **X** | | | | Puppet, es una herramienta de gestión de la configuración de código abierto [18]. | Configuración Automática de Entornos |
| *Ansible* | **X** | | | | Ansible, es una herramienta de automatización. Esta puede configurar sistemas, desplegar software y orquestar tareas más avanzadas, tales como despliegues continuos, y actualizaciones sin tiempo de indisponibilidad [18]. | Configuración Automática de Entornos |
| *Consul* | **X** | | | | Es una solución de servicios de red para automatizar configuraciones de red, soportado por las integraciones con la mayoría de herramientas de DevOps y redes [18]. | Automatización |
| *Nexus-Repository* | **X** | | | | Nexus, ofrece un repositorio administrado como una plataforma central para guardar artefactos construidos, almacenarlos, y mantener el costo del hardware [18]. | Repositorio de Artefactos |
| *Rundeck* | **X** | | | | Software de automatización open source [18]. | Virtualización de Entornos a Medida |

| | | | | | | |
|---|---|---|---|---|---|---|
| *Jenkins Pipeline Unit* | X | | | | Pipeline Unit de Jenkins es un framework de pruebas unitarias para pipelines, escrito en Groovy [18]. | Ejecución de Pruebas Unitarias |
| *CIRCL CVE Search* | | X | X | X | Cve-search, es accesible a través de una interfaz web y una API HTTP. Cve-search es una interfaz para buscar información públicamente conocida de vulnerabilidades de seguridad en software y hardware junto con sus correspondientes exposiciones [18]. | Administración Vulnerabilidades |
| *TruffleHog* | | X | X | | Busca secretos en los repositorios de git, profundizando en el historial de commits y ramas. Esto es efectivo para encontrar secretos confirmados accidentalmente [18]. | Precommit - Análisis sensible en repositorios centralizados de código |
| *Git Secrets* | | X | X | | Previene realizar un commit que contenga contraseñas u otra información sensible a un repositorio de git [18]. | |
| *GPG Signature Verification* | | X | X | | La verificación de la firma GPG en los commits y las etiquetas hace que sea fácil ver cuando un commit o una etiqueta están firmados por una clave verificada que GitHub conoce [18]. | |
| *Talisman* | | X | X | | Herramienta que instala un gancho o "hook" en el repositorio para asegurar que la información sensible y secretos no salgan de la máquina del desarrollador. Validando los cambios salientes que parecen sospechosos [92]. | |
| *Git Hooks* | | X | X | | Los ganchos o "hooks" son scripts de git útiles para identificar los errores antes y después de algunos eventos, como: confirmar, enviar y recibir [93]. | |
| *Pre Commit* | | X | X | | Framework para la administración y mantenibilidad del multi lenguajes de ganchos de pre-commit [94]. | |
| *Detect Secrets* | | X | X | | Herramienta para la detección de secretos en un código base [95]. | |
| *Git Hound* | | X | X | | Plugin de git que ayuda a prevenir que la información sensible sea enviada al repositorio centralizado [96]. | |
| *JobConfigHistory plugin* | | X | X | X | Es un plugin para Jenkins que muestra qué cambios de configuración y de trabajo son realizados y por cuál usuario [18]. | Plugin Jenkins - Detección vulnerabilidades en el pipeline configurado |
| *SCM Sync Configuration Plugin* | | X | X | X | La característica del plugin es que sincroniza los archivos de configuración con un repositorio [18]. | |
| *Audit Trail Plugin* | | X | X | X | Plugin para Jenkins es una herramienta que registra los cambios en la creación de trabajos o traza qué persona ha iniciado o eliminado la construcción [18]. | |
| *Jenkins Matrix-Based Security* | | X | X | X | Limita los privilegios del usuario y de los grupos de Jenkins [18]. | |

| | | | | | | |
|---|---|---|---|---|---|---|
| *OSSIndex* | | X | X | | OSS Index, es un catálogo gratuito de componentes de código abierto y herramientas de escaneo para ayudar a los desarrolladores a identificar las vulnerabilidades, entender el riesgo y mantener su software seguro [18]. | Plugin Análisis dependencias |
| *Retire.js* | | X | X | | La herramienta admite la detección de vulnerabilidades en las bibliotecas JavaScript utilizadas [18]. | |
| *Gradle Witness* | | X | X | | Un plugin de Gradle que permite la verificación estática de las dependencias remotas. Repositorios como MAVEN CENTRAL [18]. | |
| *Docker Hub Image Security Scan* | | X | | X | Docker Hub ha incluido un escáner de seguridad de imágenes32 que puede detectar vulnerabilidades en las imágenes oficiales de Docker [18]. | Análisis de imágenes de Docker |
| *OpenSCAP* | | X | | X | Es una solución de cumplimiento de seguridad. Con esta herramienta se pueden escanear imágenes, contenedores y máquinas virtuales [18]. | |
| *Dagda - CoreOS* | | X | | X | Las vulnerabilidades pueden detectarse en las imágenes y contenedores de Docker [18]. | |
| *Jenkins CI Image Vulnerability Scan* | | X | | X | Es una herramienta capaz de detectar vulnerabilidades en imágenes Docker construidas con CoreOS Clair en Jenkins [18]. | |
| *Banyan Collector* | | X | | X | El marco examina las imágenes [18]. | |
| *Twistlock* | | X | | X | Es una herramienta comercial capaz de detectar vulnerabilidades en los contenedores Docker [18]. | |
| *Docker Security Scanning - DSS* | | X | | X | Esta herramienta permite el análisis de seguridad a nivel binario de las imágenes del repositorio antes de que sean liberada [97]. | |
| *Nexus Repository Manager OSS Repository Health Check* | | X | X | | El Health Check analiza los componentes del repositorio y detecta sus vulnerabilidades en él [18]. | Análisis en artefactos |
| *DevSec Hardening Framework* | | X | | X | Combina DevOps con seguridad para agregar una capa de seguridad dentro de un framework de automatización, que configura sistemas operativos y servicios [18]. | Análisis en infraestructura |
| *Lynis* | | X | | X | Es una herramienta que puede detectar vulnerabilidades en sistemas Linux y basados en Unix [18]. | |
| *AWS Cloud Watch* | X | | | | Monitorea los recursos y las aplicaciones desplegadas y corriendo en AWS [18]. | Monitoreo - Retroalimentación |

| | | | | | | |
|---|---|---|---|---|---|---|
| *Elastic Logstash* | X | | | | Es una herramienta de código abierto que puede procesar y transformar los datos de diferentes tipos de fuentes [18]. | Monitoreo - Retroalimentación |
| *Sysdig Falco* | | X | | X | Es una herramienta de monitorización para la seguridad de contenedores y aplicaciones. Con esta herramienta se pueden monitorizar contenedores, aplicaciones y tráfico de red, por ejemplo, cada petición HTTP de un contenedor [18]. | Monitoreo de seguridad |
| *OWASP Security Logging Project* | | X | | X | Proyecto de registro de seguridad: es una API para el registro de eventos de seguridad [18]. | Análisis API |
| *Docker Hub/Store* | X | | | | Es la forma más fácil de crear, gestionar y entregar las aplicaciones de contenedores de sus equipos [73]. | Repositorio de imágenes de Docker |
| *CIS Benchmark* | | X | | X | Guía buenas prácticas en infraestructura [73]. | Hardening |
| *Anchore Container Image Scanner Plugin* | | X | X | X | Plugin de Anchore para Jenkins. Anchore ha sido diseñado para integrarse perfectamente en el flujo de trabajo CI/CD [74]. | Plugin Jenkins - Análisis imágenes Docker |
| *Fortify HP Static Code Analyzer* | | X | X | | Identifica las vulnerabilidades de seguridad del código fuente en una fase temprana del desarrollo de software [17]. | SAST |
| *Fortify On Demand* | | X | | | Gestione todo su programa de seguridad de las aplicaciones desde una interfaz [17]. | Administración vulnerabilidades |
| *JUnit* | X | | | | JUnit, es un conjunto de bibliotecas creadas por Erich Gamma y Kent Beck que son utilizadas en programación para hacer pruebas unitarias de aplicaciones Java [79]. | Ejecución de Pruebas Unitarias |
| *GitLab* | X | | | | Gitlab, es un servicio web de control de versiones y desarrollo de software colaborativo basado en Git. Además de gestor de repositorios, el servicio ofrece también alojamiento de wikis y un sistema de seguimiento de errores, todo ello publicado bajo una Licencia de código abierto [98]. | Repositorio Centralizado de Código |
| *Bamboo* | X | | | | Bamboo, es un servidor de integración y despliegue continuo desarrollado por Atlassian [99]. | Integrador CI/CD |
| *Buildbot* | X | | | | Framework para la integración continua [100]. | Integrador CI/CD |
| *AzureDevOps* | X | | | | Servicios DevOps ofrecidos por Microsoft [101]. | Servicio SaaS DevOps |
| *GitLab CI/CD* | X | | | | Herramienta CI/CD para desarrollar software usando metodologías continuas: CI-CD-CDe [102]. | Servicio SaaS DevOps |
| *WhiteSource* | | X | | | Validación de dependencias y manejo de código de software libre. Ahora es MEND [103]. | Análisis de dependencias |
| *CircleCI* | X | | | | Plataforma integración y entrega continua [104]. | Integrador CI/CD |

| | | | | | | |
|---|---|---|---|---|---|---|
| *SonarCloud* | **X** | **X** | **X** | | SonarCloud, es un servicio de análisis de código basado en la nube diseñado para detectar problemas de calidad del código en 25 lenguajes de programación diferentes, garantizando continuamente la mantenibilidad, fiabilidad y seguridad de su código [105]. | Verificación de Calidad del Código - SAST |
| *Subversion* | **X** | | | | Subversión es un sistema de control de versiones open source [106]. | Repositorio Centralizado de Código |
| *New Relic (APM)* | **X** | | | | Monitoreo del rendimiento de aplicaciones, ayuda a los equipos a detectar anomalías en el software [107]. | Monitoreo - Retroalimentación |
| *Apache Ant* | **X** | | | | Apache Ant, es una librería Java y una herramienta de línea de comandos cuya misión es conducir los procesos descritos en los archivos de construcción como objetivos y puntos de extensión dependientes entre sí [108]. | Gestor Paquetes/ Gestor Compilación |
| *JavaMelody* | **X** | | | | Supervisa las aplicaciones Java y Java EE tanto en entornos de control de calidad como de producción. Además, mide y calcula eficazmente las estadísticas basadas en el despliegue y el uso reales de la aplicación [109]. | Monitoreo - Retroalimentación |
| *Logstash* | **X** | | | | Logstash, es un servidor de canalización de procesamiento del lado del servidor gratuito y libre que tiene como entrada múltiples fuentes [110]. | Monitoreo - Retroalimentación |

### 3.1.3. Selección de herramientas para la plataforma DevOps y SecDevOps propuesta

Como se mencionó en el apartado de la metodología, capítulo 2.1.4, el primer criterio de selección es el lenguaje de programación Java. Para la selección del integrador y/u orquestador los criterios propuestos son: soporte de código abierto "open source", versiones pagas, alcance para los procesos CI y CD, popularidad en el mercado, facilidad de uso y buena documentación. En la Tabla 3-4 se muestran los integradores y la calificación de acuerdo con los criterios de selección.

**Tabla 3-4.** Criterios de selección para el integrador CI/CD

| **Integrador y/ u orquestador** | **Open source** | **Versiones pagas** | **CI CD** | **Popularidad (top)** [111] [112] |
|---|---|---|---|---|
| *Jenkins* | Sí | No | Ambos | 1 |
| *Teamcity* | No | Sí | Ambos | 3 |
| *Spinnaker* | Sí | No | Sólo CD | 13 |
| *Travis CI* | No | Sí | Sólo CI | 7 |
| *Concourse CI* | Sí | Sí | Ambos | / |
| *GOCD* | Sí | No | Ambos | 9 |
| *Bamboo* | No | Sí | Ambos | 4 |
| *Buildbot* | Sí | No | Sólo CI | 14 |
| *CircleCI* | No | Sí | Ambos | 2 |

El integrador que cumple con la mayoría de los criterios de selección propuestos es Jenkins. Seleccionar este orquestador implica que las demás herramientas para construir la plataforma DevOps y SecDevOps propuestas deben ser integrables con él, al igual que la disponibilidad de descarga de los plugines para permitir su integración.

Las herramientas para la plataforma DevOps propuesta correspondiente a los componentes repositorio centralizado, gestor de paquetes o de compilación, repositorio para el almacenamiento de artefactos, calidad de código, pruebas unitarias, configuración y virtualización de entorno, con sus correspondientes criterios de selección, se pueden observar en

Tabla 3-5, Tabla 3-6, Tabla 3-7, Tabla 3-8,

Tabla 3-9 y Tabla 3-10 respectivamente.

**Tabla 3-5.** Criterios de selección para el repositorio centralizado de código

| Repositorio centralizado | Open source | Versiones pagas | Integración con Jenkins | Popularidad/Descargas |
|---|---|---|---|---|
| *Github* | No | Sí | Sí | 1/228K |
| *BitBucket* | Sí | Sí | Sí | 4/ 24K |
| *GitLab* | Sí | Sí | Sí | 2/ 51K |
| *Subversion* | Sí | No | Sí | 3/ 168k |
| *Git* | Sí | No | Sí | 5/ 262k |

**Tabla 3-6.** Criterios de selección para el gestor de paquetes-compilación

| Gestor paquetes/ Gestor compilación | Integración con Jenkins | Multilenguaje (JAVA) | Descargas |
|---|---|---|---|
| *Apache Maven* | Sí, Plugin: Pipeline Maven Integration | Sí | 23K |
| *Gradle Build* | Sí | Sí | 206K |
| *Apache Ant* | Sí | Sí | 243K |

**Tabla 3-7.** Criterios de selección para el repositorio de artefactos

| Repositorio artefactos | Versión paga | Integración con Jenkins | Descargas |
|---|---|---|---|
| *JFrog Artifactory (Repositorio Universal)* | Sí | Sí, Plugin: Artifactory | 21K |
| *Nexus-repository* | Sí | Sí, Plugin: Nexus Artifact Uploader | 7K |
| *ArchiveArtifacts* | No | PostBuild | / |
| *Repositorio local Jenkins* | No | No necesita integración. Comparte el espacio Jenkins. | No es necesario descargas |

**Tabla 3-8.** Criterios de selección para el componente de verificación de calidad de código

| Verificación calidad del código | Integración con Jenkins | Multilenguaje (JAVA) | Descargas |
|---|---|---|---|

| | | | |
|---|---|---|---|
| *SonarQube Scanner* | Sí | Sí | 46K |
| *SonarCloud* | No hay plugin | Sí | / |

**Tabla 3-9.** Criterios de selección para el componente de pruebas unitarias

| Pruebas unitarias | Integración con Jenkins | Multilenguaje (JAVA) | Descargas |
|---|---|---|---|
| *JUnit* | Sí | Sí | 273K |
| *Jenkins Pipeline Unit* | Línea comando | / | / |

**Tabla 3-10.** Criterios de selección para el componente de configuración y virtualización de entornos

| Configuración/Virtualización de entornos | Open source | Versión paga | Integración con Jenkins | Descargas |
|---|---|---|---|---|
| *Docker* | Sí | Sí | Sí | 29K |
| *Azure (Cloud)* | No | Sí | Sí | |
| *Container Registry (ACR)* | No | Sí | Sí, Plugin: Azure Container Registry Tasks | 544 |
| *Kubernetes* | Sí | No | Sí | 31K |

Luego de evaluar las herramientas en relación con los criterios propuestos, en la Tabla 3-11 se presentan las herramientas para la plataforma DevOps.

**Tabla 3-11.** Herramientas seleccionadas para la plataforma DevOps

| **Herramientas elegidas para la implementación de la plataforma DevOps** | | | | | | |
|---|---|---|---|---|---|---|
| **Integrador** | **Repositorio centralizado** | **Gestor paquetes/ Gestor compilación** | **Pruebas unitarias** | **Calidad del código** | **Almacenamiento artefactos** | **Configuración/Virtualización de entornos** |
| *Jenkins* | *GitLab* | *Gradle Build* | *JUnit* | *SonarQube Scanner* | *Almacena Localmente (Jenkins)* | *Docker* |
| | | | | | | *ACR* |
| | | | | | | *Azure* |
| | | | | | | *Kubernetes* |

De igual forma, los criterios anteriormente mencionados son de uso para la selección de las herramientas de la plataforma SecDevOps. Por lo que en las tablas:

Tabla 3-12, Tabla 3-13, Tabla 3-14 y Tabla 3-15, se visualizan las herramientas para el componente SAST, análisis de dependencias, análisis de vulnerabilidades de Docker y DAST, respectivamente, con su calificación de acuerdo con los criterios de selección.

**Tabla 3-12.** Criterios de selección para el componente de SAST - CI

| SAST, Static Application Security Testing | Tipo licencia | Versiones pagas | Soporta lenguaje JAVA | Integración Jenkins | Descargas |
|---|---|---|---|---|---|
| *Checkmark* | MIT | Sí | Sí | Sí | 3.5K |
| *Veracode* | MIT | Sí | Sí | Sí | 477 |
| *ShiftLeft* | MIT | Sí | Sí | Sí, Mediante el plugin Ocular | / |
| *SonarQube* | Free &OpenSource | Sí | Sí | Sí SonarQube Scanner | 46K |
| *Warnings Next Generation* | OpenSource | No | Sí | Sí | 20K |
| *PMD* | OpenSource | No | Sí | No | / |
| *Checkstyle* | OpenSource | No | Específicamente | No | / |
| *FindBugs - SpotBugs* | OpenSource | No | Específicamente | No | / |
| *Fortify HP Static Code Analyzer* | MIT | Sí | Sí | Sí Fortify on Demand | 523 |
| *SonarCloud* | Free &OpenSource | Sí | Sí | / | / |
| *OWASP Find Security Bugs* | OpenSource | No | Sí | No | / |
| *HCL AppScan* | Comercial | Sí | Sí | Sí | 357 |

**Tabla 3-13.** Criterios de selección para el componente de análisis de dependencias - CI

| Análisis de dependencias | Tipo licencia | Versiones pagas | Soporta lenguaje JAVA | Integración Jenkins | Descargas |
|---|---|---|---|---|---|
| *Black Duck* | Free, comercial | Sí | Sí | Sí, Synopsys Detect | 1.6K |
| *Snyk* | Free, comercial | Sí | Sí | Sí | 456 |
| *OWASP Dependency Check* | OpenSource | No | Sí | Sí | 6K |
| *OSSIndex* | / | Sí | Sí | No, Se integra en el | / |

| | | | | | |
|---|---|---|---|---|---|
| | | | | plugin de Maven/Gradle | |
| *Retire.js* | / | / | No | / | / |
| *Gradle Witness* | OpenSource | No | / | No | / |
| *WhiteSource* | OpenSource | Sí | Sí | Sí | 590 |
| *SourceClear* | Free, comercial | Sí | Sí | Sí | 23 |
| *ShiftLeft* | OpenSource or Free | No | Sí | Mediante el plugin Ocular | / |
| *Dependabot* | Free | No | Sí | No | / |

**Tabla 3-14.** Criterios de selección para el componente de análisis de vulnerabilidades en imágenes Docker - CD

| Análisis de vulnerabilidades en imágenes Docker | Tipo licencia | Versiones pagas | Integración Jenkins | Descargas |
|---|---|---|---|---|
| *Clair/Klair* | Open Source | No | Sí | / |
| *Anchore Container Image Scanner* | Open Source | Sí | Sí, Anchore Container Image Scanner | 1.1K |
| *JFrog Xray* | No | Sí | Sí | / |
| *Snyk* | Free, comercial | Sí | Sí | 456 |
| *Docker Hub image security scan* | No | Sí | No | / |
| *OpenSCAP* | Open Source | No | No | / |
| *Banyan Collector* | Open Source | No | No | / |
| *Twistlock (Docker Containers)* | No | Sí | Sí | / |
| *Sysdig falco* | Open Source | Sí | Sí | 51 |

**Tabla 3-15.** Criterios de selección para el componente de DAST - CD

| DAST, Dynamic Application Security Testing | Tipo licencia | Versiones pagas | Soporta lenguaje JAVA | Integración Jenkins | Descargas |
|---|---|---|---|---|---|
| *Veracode* | MIT | Sí | Sí | Sí | 477 |
| *OWASP Zed Attack Proxy (ZAP) Spider* | Open Source | No | Sí | Sí, Official OWASP ZAP | 1.6K |
| *Crashtest Security* | No | Sí | Sí | Sí | / |
| *OWASP Purpleteam* | Open Source | No | Sí | No | / |

**Tabla 3-16.** Herramientas seleccionadas para una plataforma SecDevOps estándar - CI

| Herramientas elegidas para la implementación de la plataforma SecDevOps estándar- integración continua | |
|---|---|
| **SAST, Static Application Security Testing** | **Análisis de dependencias** |
| *SonarQube* | *OWASP dependency check* |

**Tabla 3-17.** Herramientas seleccionadas para una plataforma SecDevOps estándar - CD

| **Herramientas elegidas para la implementación de la plataforma SecDevOps estándar - despliegue continuo** | |
|---|---|
| **Análisis de vulnerabilidades en imágenes Docker** | **DAST, Dynamic Application Security Testing** |
| *Anchore Container Image Scanner* | *OWASP Zed Attack Proxy (ZAP) - Spider* |

Las herramientas seleccionadas se muestran en las Tabla 3-16 y Tabla 3-17 con su respectiva clasificación según el componente y/o estrategia de seguridad.

Es importante mencionar las siguientes herramientas con su respectiva clasificación de acuerdo con el componente y/o estrategia de seguridad adicionales para la plataforma SecDevOps propuesta. Esto se puede evidenciar en: Tabla 3-18 y Tabla 3-19.

**Tabla 3-18.** Criterios de selección para la herramienta del componente: precommit (Administración de secretos)

| **Precommit/Administración Secretos** | **Tipo licencia** | **Versiones pagas** | **Integración Jenkins o GitLab/GitHub** | **Descargas** | **Lenguaje Java soportado** |
|---|---|---|---|---|---|
| *truffleHog* | / | / | Sí (GitLab/Hub) | / | / |
| *Git secrets Secret Detection* | Free, Comercial | Sí | GitLab | / | / |
| *GPG signature verification* | Free | / | GitLab | / | / |
| *Talisman* | / | / | GitHub | / | / |
| *Git Hooks* | / | / | GitLab | / | / |
| *Pre Commit* | / | / | GitHub | / | / |
| *Detect Secrets* | / | / | GitHub | / | / |
| *Git Hound* | / | / | GitHub | / | / |
| *SonarLint** | Free, Comercial | No | IntelliJ,Eclipse/VS/VSC | / | Sí |
| *OWASP Find Security Bugs** | Free | No | IntelliJ,Eclipse | / | Sí, Java web |
| *DevSkim** | Free | No | VS/VSC | / | Sí |
| *Pumascan** | Free, Comercial | Sí | AzureDevOps/Server/EndUser | / | No, .NET C# |

**Tabla 3-19.** Criterios de selección para la herramienta del componente: Análisis de composición de software

| **SCA, Software Composition Analysis** | **Tipo licencia** | **Versiones pagas** | **Soporta lenguaje JAVA** | **Integración Jenkins** | **Descargas** |
|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| Black Duck | Free, comercial | Sí | Sí | Sí, Synopsys Detect | 1.6K |
| Snyk | Free, comercial | Sí | Sí | Sí | 456 |
| OWASP Dependency Check | OpenSource | No | Sí | Sí | 6K |
| Veracode | Free, comercial | Sí | Sí | Sí | 1.4K |
| Checkmark | MIT | Sí | Sí | Sí | 3.8K |
| Dependabot | Free, comercial | Sí | / | Sí | / |

**Tabla 3-20.** Criterios de selección para la herramienta del componente: Análisis manifiestos Kubernetes

| Análisis manifiestos Kubernetes | Tipo licencia | Versiones pagas | Integración AKS | Integración Jenkins | Descargas |
|---|---|---|---|---|---|
| Kubescape | Open source | No | Sí | Sí | / |
| Kube-hunter | Open source | No | Sí | No | / |
| KubeLinter | Open source | No | Sí – En desarrollo | No | / |
| KubeScan | Open source | No | Sí | No | / |
| Trivy | Open source | No | Sí | Sí | / |
| Kube-bench | Open source | No | Sí | No | / |

Adicionalmente, para los controles de seguridad como modelado de amenazas, verificación estándar de seguridad de aplicaciones y controles proactivos, se hace uso de la herramienta propia del proyecto OWASP.

**Tabla 3-21.** Herramientas seleccionadas para los controles de seguridad seleccionados en el proceso CI

| **Herramientas elegidas para la implementación de la plataforma SecDevOps - estrategia propuesta integración continua** | | | | |
|---|---|---|---|---|
| **Precommit / Administración Secretos** | **Modelado de Amenazas** | **ASVS, Application Security Verification Standard** | **OWASP Proactive Controls** | **SCA, Software Composition Analysis** |
| Git Secrets | OWASP ThreatDragon | OWASP ASVS | OWASP Proactive Controls | OWASP Dependency Check |
| SonarLint | | | | |

Dicho esto, en la Tabla 3-21 se consolidan las herramientas seleccionadas teniendo en cuenta su clasificación en el proceso CI y, en la Tabla 3-22, se consolida la herramienta para el proceso CD.

**Tabla 3-22.** Herramienta seleccionada para los controles de seguridad seleccionados en el proceso CD

| Análisis seguridad en kubernetes |
| --- |
| Kubescape |

## 3.2. Fase 2. Propuesta de la estrategia de seguridad en DevOps

De acuerdo al procedimiento expuesto en el capítulo 2.2.1, se plantean siete (7) parámetros que conforman una estrategia de seguridad integral y completa; clasificados en parámetros obligatorios y parámetros transversales

### 3.2.1. Selección de parámetros para el diseño de la estrategia de seguridad propuesta e intervención en los procesos de CI y CD

Los siete (7) parámetros seleccionados a partir de los criterios mencionados en el capítulo 2.2.1 constituyen una estrategia de seguridad integral, contribuyendo tanto desde el punto técnico, como en su configuración, implementación, políticas de seguridad y temas metodológicos como gestión de vulnerabilidades, metodología y temas culturales enfocados al entendimiento y adopción de la estrategia. A continuación, se mencionan los parámetros con su respectiva definición de acuerdo con su clasificación.

- ***Parámetros obligatorios en la estrategia de seguridad propuesta***

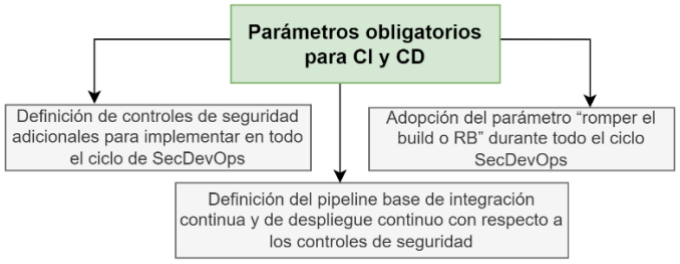



**Figura 3-12.** Parámetros obligatorios para CI y CD de la estrategia de seguridad

En la Figura 3-12 se observan los tres (3) parámetros de la estrategia de seguridad clasificados como obligatorios. A continuación, se da una resumida descripción de cada uno de estos.

A. *Definición de controles de seguridad adicionales para implementar en todo el ciclo de SecDevOps*

Se definen los controles de seguridad adicionales para implementar en todo el ciclo DevOps ayudando a incrementar el nivel de seguridad con respecto a los controles básicos de seguridad mencionados en la literatura.

B. *Adopción del parámetro romper el build durante todo el ciclo SecDevOps*

A continuación, se brinda una definición, alcance y relevancia del parámetro en la estrategia de seguridad propuesta. Presentación de la política para la evaluación del parámetro: Definición de los puntos en los procesos CI y CD y la cantidad mínima de vulnerabilidades a cumplir durante todo el ciclo DevOps.

C. *Definición del pipeline base de integración continua y de despliegue continuo con respecto a los controles de seguridad*

Estandarización y construcción de un "base-pipeline" con los respectivos pasos o "steps" de seguridad incorporando los controles de seguridad básicos y adicionales recomendados en la estrategia.

- ***Parámetros transversales en la estrategia de seguridad propuesta***

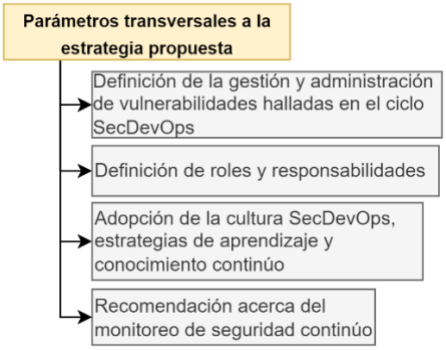



**Figura 3-13.** Parámetros transversales para CI y CD de la estrategia de seguridad

En la Figura 3-13 se observan los cuatro (4) parámetros de la estrategia de seguridad clasificados como transversales. A continuación, se da una resumida descripción de cada uno de estos.

*A. Definición de la gestión y administración de vulnerabilidades halladas en el ciclo SecDevOps*

Establecer una política para la gestión de vulnerabilidades, que facilite la clasificación y administración básica de las vulnerabilidades durante todo el ciclo DevOps.

*B. Definición de roles y responsabilidades*

Estructurar una matriz de roles y responsabilidades de cada uno de los integrantes de un equipo estándar de SecDevOps, con el propósito de mantener claras las responsabilidades y custodiar la correcta ejecución de las tareas en consecuencia con el rol establecido dentro del equipo.

*C. Adopción de la cultura SecDevOps, estrategias de aprendizaje y conocimiento continúo*

Definición de una estrategia que pueda contribuir a la adaptación fácil, ágil de la cultura SecDevOps y así mismo incentivar el aprendizaje continuo del equipo de SecDevOps.

*D. Recomendación acerca del monitoreo de seguridad continúo*

Plantear el por qué es importante tener en cuenta la implementación de una estrategia para el monitoreo de seguridad continúo y la recomendación de ciertas herramientas para implementar, extender la visibilidad y obtener la trazabilidad de seguridad durante todo el ciclo de DevOps.

### 3.2.2. Diseño de la estrategia de seguridad propuesta

A continuación, se presenta desarrollo del diseño de la estrategia de seguridad propuesta a través del desarrollo de los parámetros planteados.

- ***Parámetros obligatorios – Proceso Integración Continua (CI)***

Partiendo de la Figura 1-3, donde se visualizan las etapas ejecutadas en la integración continua, nos ubicamos en la etapa “plan”; etapa inicial del ciclo de desarrollo de software, la cual está enfocada en describir y definir las características de las aplicaciones. Por esto, se mencionan a continuación los controles adicionales planteados que aportan un nivel de seguridad complementario en esta.

*A. Definición de controles de seguridad adicionales para implementar en CI*

Los controles de seguridad propuestos en la integración continua que se van a exponer a continuación son: modelado de amenazas, controles proactivos OWASP, proyecto estándar de

verificación de seguridad de aplicaciones OWASP, precommit-administración de secretos y análisis de composición de software.

- *Modelado de amenazas*

Threat Modeling, concepto expuesto en el apartado de marco teórico, capítulo 1.1.5. Es de gran importancia realizar una comprobación de identidad y planificación de manera correcta de los posibles escenarios de riesgos y amenazas del software, desde el punto de vista de un atacante y en la fase temprana del desarrollo, esto con el fin de obtener como resultado un diseño con un panorama claro de los posibles vectores de ataques a los que pueda estar expuesto, obteniendo diagramas, requerimientos de seguridad, lista de ataques y vulnerabilidades. El enfoque proactivo ayuda a encontrar de manera más simple las amenazas, realizando una priorización basada en "bugs" y creando un plan de mitigación para el mejor entendimiento del sistema. El proceso inicia con el modelado de amenazas y la metodología para describirlo. A nivel de costos también aporta, en cuanto a que corregir vulnerabilidades en etapas tempranas demanda un costo más bajo [61] [113].

Algunas de las herramientas adicionales a la recomendada por OWASP para implementar la metodología de modelado de amenazas, son:

- PASTA – Microsoft Threat Modeling.
- OCTAVE.
- TRIKE.
- VAST [113].
- Ace Threat Analysis and Modeling (Microsoft).
- CORAS (Consultative Objective Risk Analysis System).
- PTA (Practical Threat Analysis) [61].
- Herramienta OWASP: ThreatDragon [114].

- *Controles proactivos OWASP*

OWASP Proactive Controls, concepto expuesto en el apartado de marco teórico, capítulo 1.1.5. La recomendación integral para el desarrollo de software es que, adicional a mantener las buenas prácticas, es importante contar con un punto de partida para que los desarrolladores las implementen realizando software seguro, con el fin de poder ofrecer una aplicación con seguridad desde todos sus niveles: interfaz de usuario, lógica de negocio, controlador, código de base de datos, entre otros. Por lo anterior, resulta importante mencionar los diez (10) controles proactivos, propuestos por OWASP, que deberían ser incluidos al 100% en cada proyecto. Estos describen los controles de seguridad y las categorías más importantes que deben de tener en cuenta para diseñar un software seguro.

Actualmente, hay carencia de desarrolladores con capacidades, habilidades de código seguro y pocas empresas que guíen y motiven a construirlo. Por ello se necesita proactividad desde las etapas tempranas, asumiendo la seguridad más allá de un concepto teórico, por lo que es indispensable tomar como apoyo los 10 controles que ofrece OWASP para aumentar y mejorar el desarrollo seguro.

La herramienta es recomendada por el proyecto OWASP: OWASP Proactive Controls. Versión PDF [115].

- *Proyecto estándar de verificación de seguridad de aplicaciones*

ASVS, OWASP Application Security Verification Standard, concepto expuesto en el apartado de marco teórico, capítulo 1.1.4. Aportando a las etapas de la codificación o creación del software, es necesario, además de los controles, contar con un estándar para la validación del nivel de seguridad de la aplicación que valide la implementación de esos controles. En el mismo enfoque de OWASP, es importante nombrar los tres (3) niveles que ofrece este estándar para facilitar el trabajo en el desarrollo seguro en aplicaciones y, adicionalmente, ofrecer una alineación entre necesidades, servicios de seguridad, proveedores y consumidores. Los requerimientos están desarrollados en mente para ser utilizados como métricas, guías y parámetros durante la adquisición [31] [116]. La herramienta recomendada por el proyecto OWASP es OWASP ASVS. Versión PDF [117].

- *Precommit – Administración de secretos*

Precommit, secrets management, conceptos expuestos en el apartado de marco teórico, capítulo 1.1.5. El control precommit es utilizado para encontrar y corregir errores antes de la verificación de cambios en los repositorios centralizados. Por ende, es relevante analizar el repositorio local antes de enviar código al repositorio centralizado, ya que es posible que se cuenten con brechas de seguridad debido a contraseñas, llaves de secreto, llaves de certificados e información confidencial, existentes y quemadas en el código [118] . Sumado a esto, es de gran importancia implementar controles de seguridad en el IDE del desarrollador para evitar lo explicado anteriormente. Algunas de las herramientas para implementar el análisis de secretos en el precommit y la administración de secretos:

1. Gittyleaks
2. Git-secrets
3. Repo-supervisor
4. TruffleHog
5. Git Hound
6. SonarLint

- *Análisis composición de software*

SCA, Software Composition Analysis, concepto expuesto en el apartado de marco teórico, capítulo 1.1.5. Importante en la etapa de codificación y construcción del código, ya que adicional al análisis de dependencias en el código, es necesario realizar un análisis en los componentes del código base y así prevenir riesgos como “Supply-Chain Attack”. Esta estrategia apoya a reducir los riesgos de seguridad en bibliotecas y bases de código, detecta y administra el uso de compontes de código abierto, contribuye al cumplimiento de licencias aprobadas, detección de vulnerabilidades de seguridad conocidas en los componentes y monitoreo proactivo y continuo [119].

En el apartado de resultados, *Selección de herramientas para la plataforma DevOps y SecDevOps propuesta*, capítulo 3.1.3 se exponen en: Tabla 3-18, Tabla 3-19 y Tabla 3-21, las cuales contienen tanto los criterios de selección como las herramientas para los controles de seguridad anteriormente mencionados.

### *B. Adopción del parámetro romper el build en CI*

El parámetro Romper el Build, dentro de la estrategia propuesta juega un papel fundamental a lo largo de todo el ciclo SecDevOps, ya que este nos indica si es posible continuar o no el flujo a lo largo de dicho ciclo de acuerdo a la evaluación de una política de seguridad establecida. La política de seguridad que se va a emplear en esta investigación se fundamenta en dos (2) elementos: el primero es el punto donde se desea evaluar el RB; para esta propuesta de seguridad, los lugares son al final de cada proceso de CI y de CD. Y la ruptura está definida por la cantidad mínima de vulnerabilidades que debe cumplir como referencia establecida para los riesgos de seguridad asociados al aplicativo a desarrollar y que se definen en la fase del precommit. El objetivo de este parámetro es que el software a desarrollar cumpla con los criterios de calidad con respecto a la seguridad informática prestablecidos y así evitar una salida a producción con brechas de seguridad que hagan vulnerable dicho aplicativo.

En la Tabla 3-23 se menciona la clasificación de las vulnerabilidades basada en OWASP, tomada como insumo apropiado para la construcción de la política establecida desde la estrategia de seguridad propuesta.

**Tabla 3-23.** Clasificación vulnerabilidades

| Clasificación Vulnerabilidades |
|---|
| Críticas |
| Altas |
| Medias |
| Bajas |
| Informativas |

La cantidad de vulnerabilidades propuestas permitidas por el parámetro RB pueden ser adaptables a la madurez del equipo de seguridad de la empresa (tamaño de empresa), la valoración de los riesgos vinculados a la aplicación en desarrollo y al nivel de riesgo que se quiera asumir. En la Tabla 3-24 se expone un ejemplo de la cantidad mínima de vulnerabilidades permitidas por la evaluación del parámetro RB.

**Tabla 3-24.** Propuesta de la cantidad de vulnerabilidades permitidas por el parámetro RB

| Clasificación Vulnerabilidades | Cantidad |
|---|---|
| Críticas | 1 |
| Altas | 3 |

| | |
|---|---|
| Medias | 4 |
| Bajas | 5 |
| Informativas | Cualquiera |

Adicionalmente, en la Figura 3-14 se observan los controles de seguridad en el ciclo SecDevOps mencionados en la literatura y los controles planteados desde la estrategia de seguridad propuesta. Basados en esto, se menciona el punto en el proceso CI como segunda directriz de la política para la evaluación del RB.

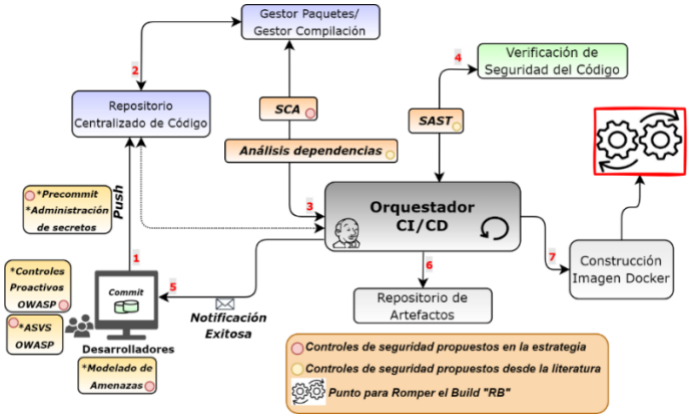



**Figura 3-14.** Puntos propuestos para evaluar el parámetro RB – CI

*C. Definición del pipeline base de integración continua*

Los pasos o steps construidos en el pipeline para la ejecución de los controles de seguridad propuestos y los controles estándares se visualizan a continuación. El código expuesto está basado en lenguaje groovy y se incluye en un archivo llamado "Jenkinsfile" el cual es interpretado por el orquestador Jenkins sin ningún inconveniente. En este código se configuran las diferentes variables de entorno necesarias para la ejecución, el compilador para el código y la versión del JDK de JAVA usada por Jenkins para realizar la ejecución del pipeline. Luego de esto compila el código y ejecuta las pruebas unitarias desarrolladas, inicia con los pasos o steps de seguridad propuestos por la estrategia de seguridad: análisis de dependencias, análisis de código estático y se construye la imagen docker con el ejecutable generado. En cada step se genera el reporte con los hallazgos encontrados, al final se debe evaluar el parámetro RB de acuerdo a la cantidad de vulnerabilidades

propuestas en la Tabla 3-24 sobre el consolidado de los hallazgos, de acuerdo al resultado de la evaluación se continua o no con el ciclo SecDevOps.

```
pipeline {
 agent any
 environment {
  url_target= "http://20.97.136.205"
  registryCredential= "ACR"
  registryURL = "myacrsecdevops.azurecr.io"
  registryName = "base-backend-java"
  dockerImage= ""
 }
 options {
  buildDiscarder(logRotator(numToKeepStr: '3'))
            disableConcurrentBuilds()
 }
 tools {
  jdk 'JDK11'
  gradle 'Gradle5.6'
 }
 stages{
  stage('Checkout') {
   steps{
    echo "------------>Checkout SCM<------------"
                                    checkout scm
   }
  }
//CONTINUOUS INTEGRATION
  stage('Compile & Unit Tests') {
   steps{
    echo "------------>Compile & Unit Tests<------------"
    sh 'gradle --b ./restaurante/build.gradle clean test'
   }
  }
  stage('Build') {
   steps {
    echo "------------>Build<------------"
                        sh 'gradle --b ./restaurante/build.gradle build -x test'
   }
  }
  stage('Dependency Analysis - SCA') {
   steps {
    echo "----->Dependency Analysis - SCA<-------"
     dependencyCheck additionalArguments: '''
      --out "./"
      --scan "./restaurante/build/libs/*.jar"
      --format "XML"
      --log "./log-AD.log"
      --prettyPrint ''', odcInstallation: 'Dependency-Check'
      dependencyCheckPublisher pattern: 'dependency-check-report.xml'
   }
  }
// Se genera el reporte con los hallazgos de: Análisis de dependencias/Análisis composición de código.
  stage('Static Code Analysis') {
   steps{
     echo '------------>SAST Analysis<------------'
     withSonarQubeEnv('SonarQube') {
      sh "${tool name: 'sonar_scanner2', type:'hudson.plugins.sonar.SonarRunnerInstallation'}/bin/sonar-scanner -
Dsonar.projectKey=co.com.tesis.devsecops.poc.${BRANCH_NAME} -Dsonar.projectName=Tesis-DevSecOpsPoC.${BRANCH_NAME} -
Dproject.settings=sonar-project.properties"
     }
```

```
    }
   }
// Se genera el reporte con los hallazgos de: Análisis de código estático - SAST
   stage('Build Docker Image') {
    steps{
     echo "------------>Build Docker Image<------------"
     script{
       dockerImage = docker.build registryName
     }
    }
```

***//EVALUACIÓN DEL PARÁMETRO RB***

**Figura 3-15.** Pipeline base para CI

- ***Parámetros obligatorios – Proceso Despliegue Continuo (CD)***

A continuación, se mencionan los controles de seguridad recomendados para incrementar la seguridad en el proceso de despliegue continuo en el ciclo SecDevOps.

*A. Definición de controles de seguridad adicionales para implementar en CD*

- *Análisis de seguridad en kubernetes*

Dado que la infraestructura empleada para el despliegue del aplicativo web es Kubernetes es necesario realizar un escaneo sobre dicho clúster para así tener un control de seguridad sobre las posibles vulnerabilidades desencadenadas por malas configuraciones en este.

*B. Adopción del parámetro romper el build en CD*

La definición expuesta y detallada previamente fue enfocada a la integración continua, la cual también aplica para el despliegue continuo ajustando el punto donde se debe evaluar el RB en este proceso, este punto se observa en la Figura 3-16.

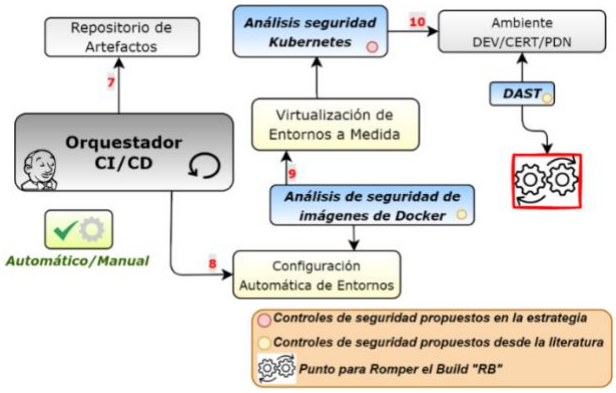



**Figura 3-16.** Puntos propuestos para evaluar el parámetro RB – CD

### *C. Definición del pipeline base de despliegue continuo*

Los pasos o steps construidos en el pipeline para la ejecución de los controles de seguridad propuestos y estándares se visualizan a continuación. Con respecto al pipeline CI, este código ejecuta otros steps relacionado al proceso CD: escaneo de seguridad en el contenedor generado en CI, carga de la imagen docker al ACR, despliegue en AKS y ejecución del análisis dinámico.

```
pipeline {
 agent any
 environment {
  url_target= "http://20.97.136.205"
  registryCredential= "ACR"
  registryURL = "myacrsecdevops.azurecr.io"
  registryName = "base-backend-java"
  dockerImage= ""
 }
 options {
  buildDiscarder(logRotator(numToKeepStr: '3'))
            disableConcurrentBuilds() }
 tools {
  jdk 'JDK11'
  gradle 'Gradle5.6'
 }
 stages{
  stage('Checkout') {
   steps{
    echo "------------>Checkout SCM<------------"
                                    checkout scm
   }
  }
//CONTINUOUS DEPLOYMENT
  stage('Container Security Scan') {
   steps {
    echo "------------>Anchore Scannig<-------------"
    sh 'echo "${registryURL}/${registryName}:latest `pwd`/Dockerfile" > anchore_images'
    anchore name: 'anchore_images', bailOnFail: "false", forceAnalyze: "true"
   }
  }
// Se genera el reporte con los hallazgos de: Análisis de seguridad en contenedores
  stage('Upload Image to ACR') {
   steps{
    echo "------------>Upload Image to ACR<------------"
    script{
     docker.withRegistry("http://${registryURL}", registryCredential){
      dockerImage.push("latest")
      dockerImage.push("${env.BUILD_NUMBER}")
     }
    }
   }
  }
  stage('Deploy on AKS') {
   steps{
    echo "------------>Deploy on AKS<------------"
    withKubeConfig([credentialsId:          'AKS-Config',          serverUrl:          'https://aks-secdevops-prod-eastus2-dns-
7765a616.hcp.eastus2.azmk8s.io:443']) {
     sh 'kubectl apply -f ./k8s-deployment.yaml'
     sh 'kubectl get pods -n ns-backend'
    }
```

```
    }
   }
   stage('Dynamic Code Scanning') {
    steps {
     echo "------------>DAST Analysis<------------"
     script  {
      def data = "sudo docker stop owasp2\nsudo docker rm owasp2\nsudo docker pull owasp/zap2docker-stable\nsudo docker run -dt
--name owasp2 owasp/zap2docker-stable /bin/bash\nsudo docker exec owasp2 mkdir /zap/wrk\nsudo docker exec owasp2 zap-full-
scan.py -t ${url_target} -r report-full-scan-DAST.html -a -d -j -I\nsudo docker cp owasp2:/zap/wrk/report-full-scan-DAST.html
/home/admin_devsecops/dast-report/report-full-scan-DAST.html\n"
      def remote = [:]
      remote.name = 'Zap-Server'
      remote.host = '192.16.2.4'
      remote.user = 'admin_devsecops'
      remote.password = '@@Admin_DevSecOps**'
      remote.allowAnyHosts = true
      writeFile file: 'baseline-scan-zap.sh', text: data
      sshScript remote: remote, script: "baseline-scan-zap.sh"
     }
    }
// Se genera el reporte con los hallazgos de: Análisis dinámico de seguridad - DAST
   }
  }
}
//EVALUACIÓN DEL PARÁMETRO RB
```

**Figura 3-17.** Pipeline base para CD

- ***Parámetros transversales para el proceso de CI y CD***

*A. Definición de la gestión y administración de vulnerabilidades halladas en el ciclo SecDevOps*

La gestión de vulnerabilidades continua permite de forma constante identificar y resolver problemas potenciales, que tradicionalmente deben esperar para ser trabajados durante ciclos periódicos. Su objetivo es alertar en tiempo real la afectación sobre los sistemas y demás componentes evaluados, activando tareas que permitan dar una visibilidad y ofrecer una trazabilidad para actuar de manera proactiva en el menor tiempo posible [120]. Puede ser un proceso complejo, ya que depende de ciertos factores críticos como: escasez de personal dedicado y especialista en seguridad, deuda técnica en la implementación, falta de comunicación con las demás áreas para la integración de los procesos, bajo presupuesto destinado para la ejecución y/o adquisición de una solución y poca madurez del equipo de seguridad y de la compañía. Cabe mencionar las ventajas que ofrece una buena gestión de vulnerabilidades: identificación de fallas complejas, continuidad de negocio, incremento de la fiabilidad e integridad de los datos; proporciona un mayor control de seguridad y contribuye a la ejecución y cumplimiento de políticas de seguridad establecidas. Dicho lo anterior y partiendo del concepto dado por la literatura de SecDevOps de implementar seguridad desde las etapas tempraneras hasta el despliegue del software, es pertinente realizar un esfuerzo en la adopción de la gestión de vulnerabilidades

durante todo el ciclo SecDevOps, para así disponer de un panorama general del comportamiento de seguridad del software y llevar a cabo una trazabilidad para un control efectivo y proactivo de las vulnerabilidades.

*B. Definición de roles y responsabilidades*

Es importante mencionar que, para la conformación del equipo de seguridad y la definición de los roles y responsabilidades, es necesario definir el alcance de las compañías al cual se va a enfocar. Cabe aclarar que un factor relevante e influyente para este punto es la madurez de la compañía en temas de seguridad y el presupuesto que puedan inyectar en proyectos relacionados. La categorización definida se enfoca en empresas pequeñas y medianas: Pymes y grandes empresas.

Como estrategia de seguridad adaptable a la madurez de la compañía, presupuesto y cantidad de personas en el equipo, se propone un grupo de trabajo para llevar acabo una implementación mínima de seguridad. Primeramente, se debe introducir el concepto de la matriz RACI que nos ayudará con la construcción de la matriz de roles y responsabilidades designadas al equipo propuesto [121]. A continuación, en la Tabla 3-25, se explica la nomenclatura de la matriz RACI.

**Tabla 3-25.** Siglas Matriz RACI. Adaptada de **[121]**

| Sigla | Traducción inglés | Traducción español |
|---|---|---|
| **R** | Responsible | Comprometido |
| **A** | Accountable | Responsable |
| **C** | Consulted | Consultado |
| **I** | Informed | Informado |

De acuerdo a lo mencionado anteriormente, a continuación, se presenta la matriz RACI para dos tipos de entidades: pymes y grandes empresas (ejemplo hipotético), con su respectivo equipo de trabajo el cual está sujeto al presupuesto y madurez de la entidad a la que aplique.

- *Pymes (Pequeñas o medianas empresas)*

El equipo estándar de seguridad en DevOps recomendado se presenta a continuación:

  - Líder/Analista SecDevOps.
  - Desarrollador con conocimientos en desarrollo seguro.

Tareas o responsabilidades ejecutadas por los integrantes del equipo de SecDevOps

1. Implementación de nuevas estrategias de seguridad en todo el ciclo SecDevOps.
2. Administración y soporte en CI y CD.
3. Adopción cultura SecDevOps y definición de nuevas políticas dentro del equipo.
4. Codificación segura.
5. Auditorías del proceso de seguridad en DevOps implementado.

**Tabla 3-26.** Matriz RACI propuesta equipo para una empresa pequeña o pyme

| Cargo/Responsabilidades | Líder/Analista SecDevOps | Desarrollador |
|---|---|---|
| 1 | I | R |
| 2 | I | R |
| 3 | R | R |
| 4 | R | R |
| 5 | R | I |

Para este tipo de empresas se recomienda el número mínimo de vulnerabilidades permitidas expuesto en la Tabla 3-24.

- *Grandes empresas*

El equipo estándar de seguridad en DevOps recomendado se menciona a continuación:

- Líder SecDevOps.
- Analista SecDevOps con profundización en integración continua.
- Analista SecDevOps con profundización en despliegue continuo.
- (2) Desarrollador con conocimientos o habilidades en desarrollo seguro.

Tareas o responsabilidades ejecutadas por los integrantes del equipo de SecDevOps

1. Implementación de nuevas estrategias de seguridad en CI.
2. Implementación de nuevas estrategias de seguridad en CD.
3. Administración y soporte en CI.
4. Administración y soporte en CD.

5. Mantenimiento y adopción de la cultura SecDevOps dentro del equipo.
6. Codificación segura.
7. Auditorías del proceso de seguridad en DevOps implementado.
8. Definición de nuevas políticas de seguridad dentro del equipo de SecDevOps.

**Tabla 3-27.** Matriz RACI propuesta equipo para una grande empresa

| Cargo/Responsabilidades | Líder SecDevOps | Analista SecDevOps CI | Analista SecDevOps CD | Desarrollador |
|---|---|---|---|---|
| 1 | I | R | C | C |
| 2 | I | C | R | C |
| 3 | I | R | A | R |
| 4 | I | A | R | C |
| 5 | R | A | A | A |
| 6 | I | A | C | R |
| 7 | R | C | C | C |
| 8 | R | C | C | C |

Luego de esto, se expone a continuación la propuesta de cantidad de vulnerabilidades para una empresa grande, la cual puede contar con un mayor presupuesto para invertir en temas de seguridad y así mismo conformar un buen equipo de seguridad. Ver Tabla **3-28**.

**Tabla 3-28.**Propuesta de la cantidad de vulnerabilidades permitidas por el parámetro RB para una empresa grande

| Clasificación Vulnerabilidades | Cantidad |
|---|---|
| Críticas | 0 |
| Altas | 2 |
| Medias | 3 |
| Bajas | 4 |
| Informativas | Cualquiera |

*C. Adopción de la cultura SecDevOps, estrategias de aprendizaje y conocimiento continúo*

Más allá de lo técnico, herramientas, procesos para la implementación y entrega de software rápida y frecuente, SecDevOps es cuestión de cultura, donde las personas son un eslabón importante en hacerlo posible. Al integrar seguridad en el ciclo de DevOps es importante crear una cultura de responsabilidad compartida y no buscar culpables al momento del error, cualquiera que esté trabajando en SecDevOps debería tener la capacidad de indicar un problema de seguridad, socializarlo con su nivel de relevancia e implementar una posible solución [122]. Basados en esto, se debe procurar lograr una comunicación asertiva en todo el equipo, tener claridad sobre roles y

responsabilidades y lograr la adopción de una mentalidad de cambio continuo y responsabilidad de la seguridad en el software.

En paralelo a la adopción de una cultura SecDevOps, también se mencionan algunas estrategias para apoyar el aprendizaje continuo en el equipo SecDevOps:

- Creación de un plan de capacitación o “Learning-Paths” para ser desarrollados y ejecutados por los integrantes del equipo de SecDevOps.
- Capacitaciones, entrenamientos, workshops de proveedores de seguridad.
- Capacitaciones para la adopción de las tecnologías usadas en SecDevOps.
- Capacitaciones para adoptar el uso adecuado de herramientas corporativas.

*D. Recomendación acerca del monitoreo de seguridad continúo*

En la actualidad existe una variedad de categorías de monitoreo, entre ellas: monitoreo de aplicación, monitoreo en el rendimiento de la infraestructura y monitoreo de seguridad. Cualquiera que sea la categoría a implementar, es necesario disponer de una estrategia que esté delimitada a un alcance para la facilitar la selección de una o dos herramientas que puedan cumplir con la necesidad requerida; adicionalmente, se aconseja examinar si las herramientas nativas pueden cumplir con los requisitos mínimos para ser integradas como herramienta de monitoreo y así aprovechar la convergencia con la tecnología de la aplicación desplegada [123]. Dado esto, es importante tener en cuenta que el monitoreo proactivo funciona como un mecanismo de alerta temprana, en contraste con el enfoque tradicional de monitorear solamente durante las últimas etapas en producción, ofreciendo una visión continua de la trazabilidad a lo largo de todo el flujo establecido. Por lo tanto, se convierte en un punto fundamental hacer la elección de un monitoreo continuo durante todo el ciclo de DevOps. A continuación, se menciona una de las herramientas para monitoreo continuo de seguridad pagas: Hdiv Security. Donde esta, es una herramienta encargada en monitorear el comportamiento de aplicaciones con la capacidad de proporcionar una detección precisa de vulnerabilidades conocidas y no conocidas “Zero-Day” en tiempo de ejecución. Esto permite a los diferentes desarrolladores evaluar, rastrear y monitorear el riesgo al ejecutar sus aplicaciones [124].

Para finalizar con el desarrollo de los parámetros propuestos, se muestra en la Figura 3-18 el diagrama de flujo de estos a lo largo del proceso CI y CD del ciclo DevOps.

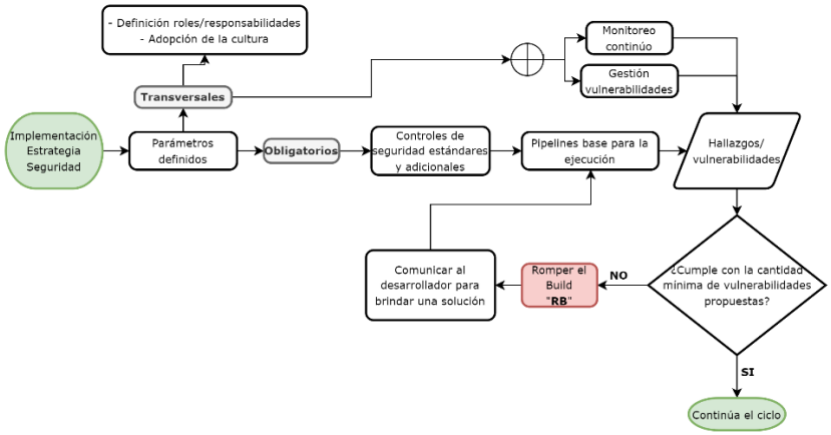



**Figura 3-18.** Flujo parámetros de la estrategia de seguridad propuesta

## 3.3. Fase 3. Implementación de la estrategia de seguridad propuesta en SecDevOps

A continuación, se presentan los resultados obtenidos al aplicar el procedimiento expuesto en la metodología.

### 3.3.1. Diseño de la plataforma de DevOps y SecDevOps

- ***Diseño de la plataforma DevOps***

Partiendo de la selección de cada una de las herramientas que componen la plataforma DevOps, expuestas en la Tabla 3-11, capítulo 3.1. Se ilustra en la Figura 3-19 la relación de cada una de estas con su respectivo componente.

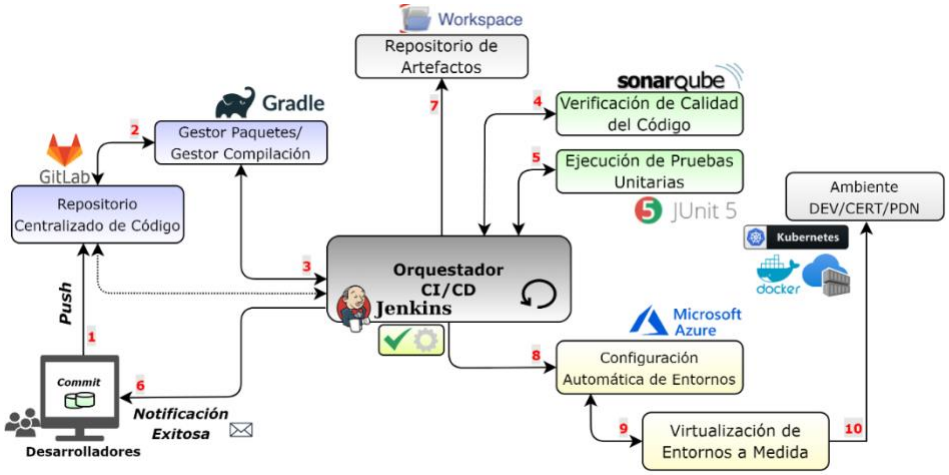



**Figura 3-19**. Componentes de la plataforma DevOps con sus respectivas herramientas

El componente principal de la plataforma DevOps es el orquestador CI/CD Jenkins, el cual cumple con la función de automatizar los procesos de integración continua y despliegue continuo a partir del despliegue de cada una de las herramientas de la Figura 3-19. Apoyados en la descripción de la plataforma DevOps dada en el capítulo 2.3.1, se muestra en la Figura 3-20 el diseño de la plataforma DevOps.

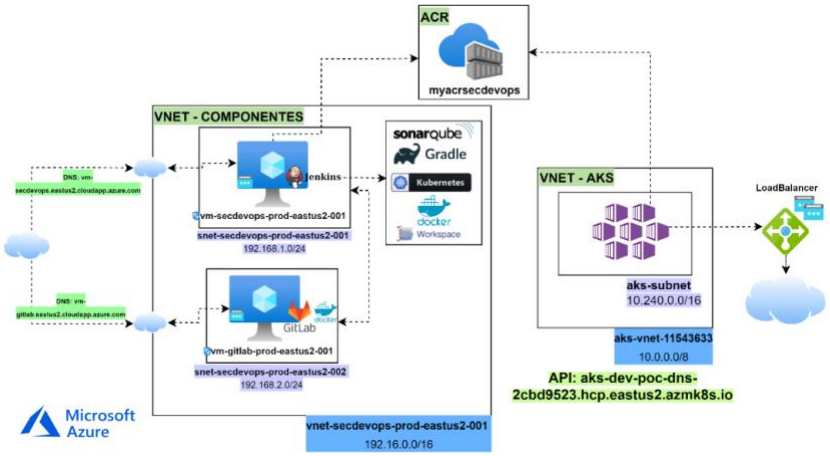



**Figura 3-20**. Diseño plataforma DevOps

- ***Diseño de la plataforma SecDevOps***

Teniendo como base los componentes de la plataforma DevOps desplegados, se procedió a instalar las herramientas adicionales para construir la plataforma DevSecOps. Para la descripción de esta se divide en los componentes que interactúan en el proceso CI y proceso CD, respectivamente. A continuación, en la Figura 3-21 se muestran las herramientas de seguridad instaladas para el proceso de integración continua. Para comprobar el nivel de seguridad en las dependencias y el

software libre del código, se instaló el plugin Dependency-Check recomendado por OWASP. La herramienta SonarQube instalada previamente, se empleó para ejecutar el análisis estático de código en lo que corresponde a seguridad.

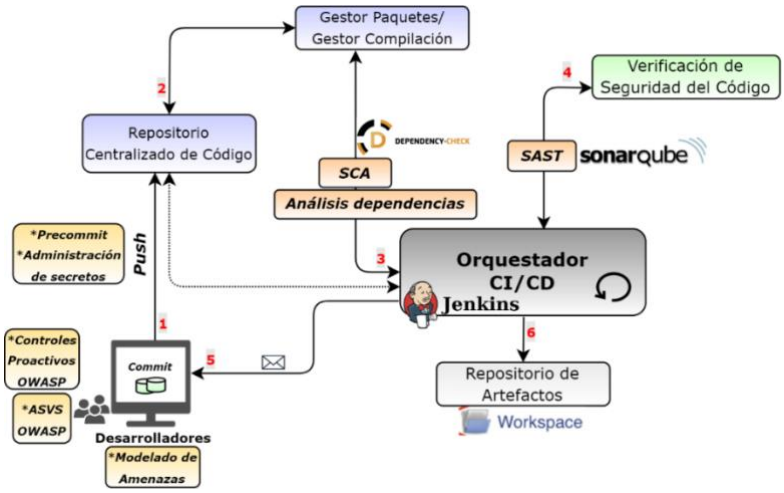



**Figura 3-21**. Componentes de la plataforma SecDevOps - CI con sus respectivas herramientas

Respecto de los componentes que intervienen en el proceso de CD, se muestran en la Figura 3-22 las herramientas correspondientes.

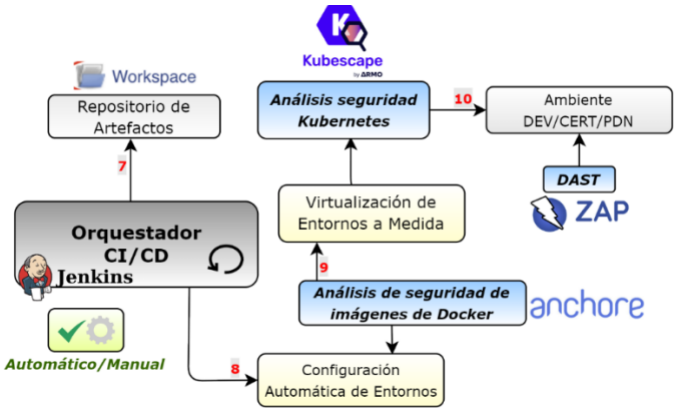



**Figura 3-22.** Componentes de la plataforma SecDevOps - CD con sus respectivas herramientas

Luego de construir la imagen de Docker, se ejecuta un análisis de seguridad con la herramienta Anchore, la cual identifica si hay brechas de seguridad a nivel de dependencias, permisos elevados o malas configuraciones en la imagen construida. Dado que el ambiente para el despliegue del aplicativo está basado en Kubernetes, se configuró la herramienta Kubescape para evaluar el nivel de seguridad en la configuración del clúster de Kubernetes y los manifiestos usados para el

despliegue de la aplicación. Para finalizar, se ejecutó un análisis dinámico y/o análisis de caja negra sobre la URL del endpoint expuesto para consumir el aplicativo.

Apoyados en la descripción de cada una de las herramientas que componen la plataforma SecDevOps, se muestra en la Figura 3-23 el diseño de esta.

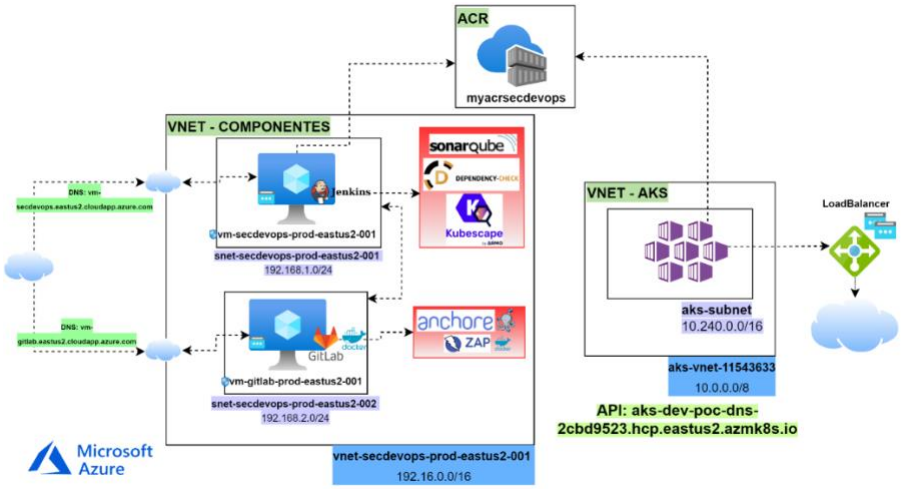



**Figura 3-23.** Diseño plataforma SecDevOps

## 3.3.2. Implementación de las plataformas DevOps y SecDevOps

La implementación de las plataformas está basada en componentes de infraestructura ofrecidos por Azure. Se desplegaron dos (2) máquinas virtuales donde se configuraron las herramientas seleccionadas en el capítulo 3.1.3. Adicional a esto, se hizo uso del componente, registro de contenedores de Azure (Azure Container Registry, ACR) donde se almacenaron las imágenes Docker y del servicio administrado de kubernetes (Azure Kubernetes Service, AKS) como ambiente para el despliegue del software.

A nivel de networking, se configuró una red virtual (Virtual Network, VNET) con dos subredes (Subnets), una para la máquina virtual donde se ejecuta el orquestador Jenkins y la otra para la máquina donde se configuró el repositorio centralizado GitLab. La comunicación de dichas máquinas se realiza internamente mediante direccionamiento IP privado. Adicionalmente, en temas de administración de cada una de las máquinas virtuales, se configuró el acceso mediante grupos de seguridad (Network Security Groups, NSG).

A partir de la construcción del diseño mencionado en el capítulo 3.3.1, se presenta el paso a paso seguido para el despliegue de cada una de las herramientas.

- ***Implementación plataforma DevOps***

La implementación e integración de cada una de las herramientas: Jenkins, GitLab, Gradle, JUnit, SonarQube, Docker, ACR y Kubernetes se presentan en el *Anexo: Instalación de las herramientas para la plataforma DevOps propuesta.*

- ***Implementación plataforma SecDevOps***

De acuerdo con la estrategia de seguridad desarrollada en el capítulo 3.2, se proponen dos puntos donde se plantea romper el build (RB). El primer punto planteado es ubicado al finalizar la construcción de la imagen Docker; en este punto del flujo, la persona de seguridad encargada consolida los hallazgos encontrados luego de ejecutar las estrategias de seguridad: análisis de dependencias, análisis de composición de software y análisis de código estático, y realiza un chequeo a cada uno de los hallazgos encontrados para determinar si corresponde a una vulnerabilidad o a un falso positivo. Luego del chequeo, se construye el informe final especificando cuáles son vulnerabilidades o falsos positivos en el proceso CI. Si el resultado del informe final contiene vulnerabilidades y no cumple con la política descrita en la estrategia, se debe romper el build y, por ende, las vulnerabilidades halladas se deben remediar para continuar el flujo; por otro lado, si el resultado cumple con la política propuesta, el flujo continúa. En la Figura 3-24 se expone lo explicado anteriormente.

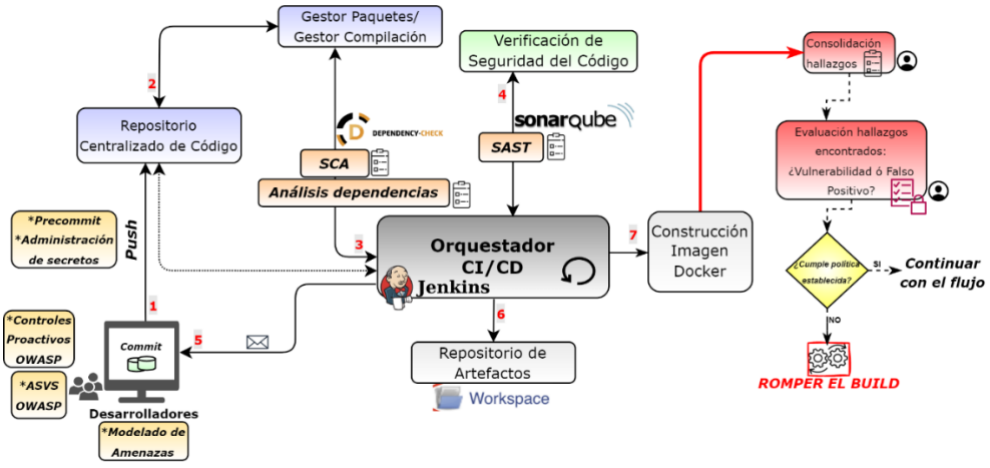



**Figura 3-24.** Romper el Build "RB" en CI

El segundo punto propuesto para ser evaluado el parámetro RB es al finalizar el despliegue del aplicativo en ambientes pre-productivos. Igual que en el punto explicado anteriormente, la persona

encargada consolida los hallazgos encontrados luego de ejecutar las estrategias de seguridad: análisis de seguridad de imágenes de Docker, análisis al entorno de Kubernetes y análisis de código dinámico. Se evalúa la política en relación con las vulnerabilidades confirmadas y se toma la decisión de romper o no romper el build (RB), en la Figura 3-25 se observa el detalle para el CD.

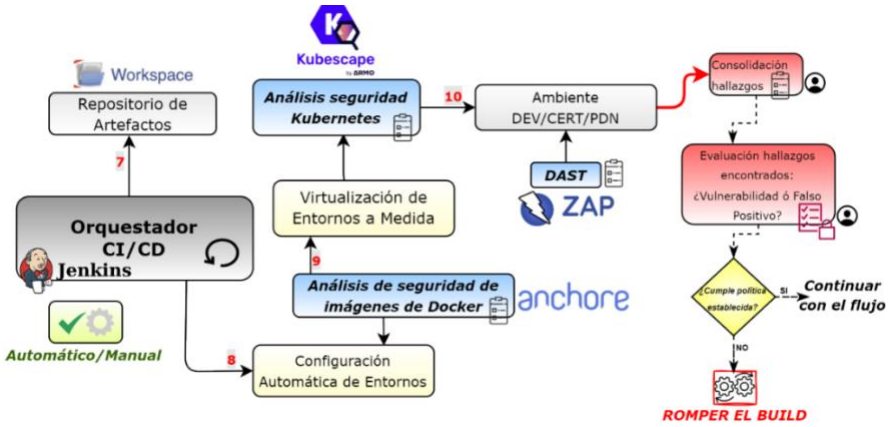



**Figura 3-25.** Romper el Build "RB" en CD

Partiendo de la plataforma DevOps implementada, el despliegue e integración de cada una de las herramientas expuestas previamente: SonarQube, OWASP Dependency-Check, Anchore, Kubescape y ZAP, se presentan en el *Anexo: Instalación de las herramientas para la plataforma SecDevOps propuesta*.

## 3.4. Fase 4. Validación y verificación de la estrategia de seguridad propuesta en SecDevOps

A continuación, se presentan los resultados obtenidos al aplicar el procedimiento expuesto en la metodología.

### 3.4.1. Definición aplicativo web prueba

Orientados por los criterios propuestos en la metodología, el aplicativo seleccionado es de uso empresarial, es decir, cumple con las buenas prácticas de desarrollo de software establecidas por la compañía y desplegado en ambientes pre-productivos para su certificación y posterior puesta en producción. El aplicativo está construido bajo una arquitectura hexagonal, compuesto por las capas de aplicación, dominio e infraestructura. Dicho software fue facilitado por la compañía Ceiba Software para la ejecución de pruebas durante el desarrollo de esta tesis, por esta razón, se debe

cumplir con la confidencialidad de código fuente. El aplicativo fue desarrollado para un restaurante, el cual cumple con las funcionalidades de llevar un control sobre los clientes, platos registrados, usuarios y ventas realizadas. Cada una de estas capacidades se ve reflejada en un microservicio expuesto por endpoints públicos para su consumo. Luego de la descripción expuesta previamente, a continuación, se mencionan los requerimientos funcionales del software:

- El sistema permitirá chequeo de salud de los servicios expuestos.
- El sistema permitirá listar el cliente y el usuario registrado en la base de datos.
- El sistema permitirá listar el cliente registrado que más ha comprado en el restaurante.
- El sistema permitirá crear un cliente y un usuario para almacenarlo en la base de datos.
- El sistema permitirá listar los platos que actualmente se ofrecen en el restaurante.
- El sistema permitirá actualizar y eliminar en cualquier momento un usuario.
- El sistema permitirá crear la venta con un identificador único y la relaciona al usuario o cliente previamente creado.
- El sistema permitirá buscar un plato específico para revisar sí aún está disponible dentro del stock del restaurante.
- Como tareas de administrador del restaurante, el sistema ofrecerá la funcionalidad de listar todas las ventas registradas al momento. Y adicionalmente, visualizar las ventas por un cliente específico.
- El sistema permitirá listar el plato más vendido en un período de tiempo.

Basados en lo anterior, se expone el diagrama de casos de uso en la Figura 3-26.

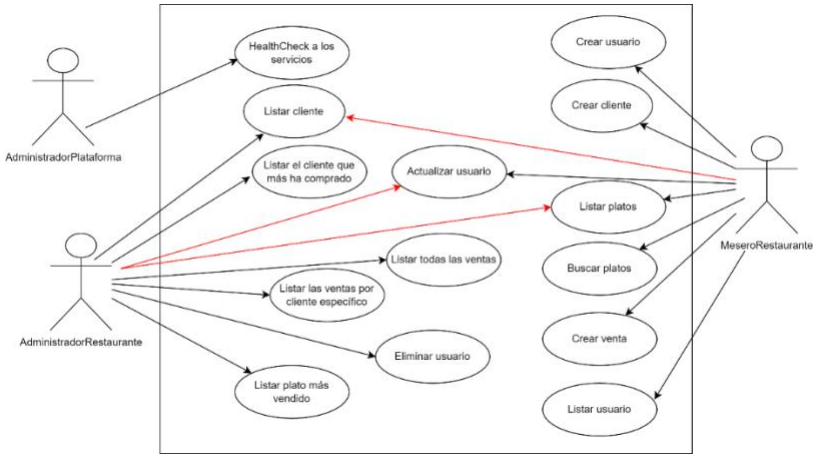



**Figura 3-26.** Diagrama casos de uso

A nivel de requerimientos no funcionales, a continuación, se mencionan las especificaciones técnicas del software tomadas dentro de los criterios de definición:

- Framework para la construcción de código de licencia libre: Spring Boot 2.1.2.
- Modelo de base de datos basado en MySQL y uso de la herramienta licencia libre: Flayway para administrar todos los scripts DDL e inicializadores de la base de datos.
- Acceso a la base de datos por medio de plantilla JDBC.
- Uso de Java 8.
- El IDE debe tener configurado Lombok para la facilitar ciertas ventajas a la hora del desarrollo.
- Gradle 5.0 para la construcción del código.

### 3.4.2. Ejecución de pruebas sobre las plataformas implementadas

Seguidamente, se exponen los resultados obtenidos de cada una de las tres pruebas propuestas para validar la estrategia de seguridad desarrollada a lo largo de la tesis.

- ***Prueba 1. Aplicativo web ejecutado y desplegado sobre la plataforma DevOps***

El procedimiento llevado acabo para la construcción, ejecución y despliegue del aplicativo bajo la plataforma DevOps se menciona a continuación.

A partir del proyecto “SecDevOps_Project” creado en GitLab, se configuró la rama “feature_prueba_devops” con el fin de manejar la ramificación dentro del proyecto original y

realizar los cambios necesarios sin interferir en las otras ramas. Luego de esto, se exponen en la Figura 3-27 los recursos del repositorio necesarios para llevar acabo la ejecución de la prueba 1.

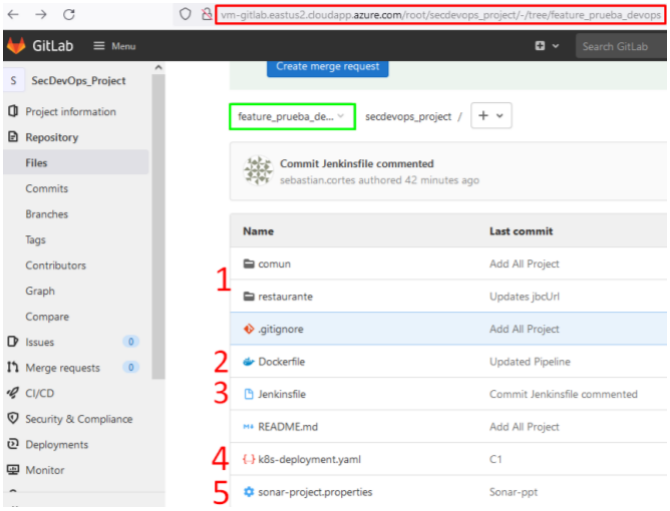



**Figura 3-27.** Recursos del repositorio usado para la prueba 1

A continuación, se presenta una breve explicación de cada uno de los recursos mencionados en la Figura 3-27.

1. Los directorios “común” y “restaurante” corresponden al código fuente del aplicativo web.
2. Dockerfile: archivo que contiene las instrucciones necesarias para la creación de la imagen Docker. En la Figura 3-28 se observa el archivo Dockerfile usado para la prueba 1.


**Figura 3-28.** Detalle Dockerfile

3. Jenkinsfile: archivo que contiene el paso a paso de las tareas que se ejecutaron en el pipeline.

```
pipeline {
 agent any
 environment {
  registryCredential= "ACR"
  registryURL = "myacrsecdevops.azurecr.io"
  registryName = "base-backend-java"
  dockerImage= ""
 }
 options {
```

```
  buildDiscarder(logRotator(numToKeepStr: '3'))
                disableConcurrentBuilds()
 }
 tools {
  jdk 'JDK11'
  gradle 'Gradle5.6'
 }
 stages{
  stage('Checkout') {
   steps{
    echo "------------>Checkout<------------"
                                        checkout scm
   }
  }
  stage('Compile & Unit Tests') {
   steps{
    echo "------------>Compile & Unit Tests<------------"
    sh 'gradle --b ./restaurante/build.gradle clean test'
   }
  }
  stage('Static Code Analysis') {
   steps{
      echo '------------>Análisis de código estático<------------'
      withSonarQubeEnv('SonarQube') {
       sh "${tool name: 'sonar_scanner2', type:'hudson.plugins.sonar.SonarRunnerInstallation'}/bin/sonar-scanner -
Dsonar.projectKey=co.com.tesis.devsecops.poc.${BRANCH_NAME} -Dsonar.projectName=Tesis-DevSecOpsPoC.${BRANCH_NAME} -
Dproject.settings=sonar-project.properties"
      }
    }
  }
  stage('Quality Gate') {
   steps {
    timeout(time: 1, unit: 'MINUTES') {
     sleep 30
     waitForQualityGate abortPipeline: true
    }
   }
  }
  stage('Build') {
   steps {
    echo "------------>Build<------------"
                          sh 'gradle --b ./restaurante/build.gradle build -x test'
   }
  }
  stage('Build Docker Image') {
   steps{
    script{
     dockerImage = docker.build registryName
    }
   }
  }
  stage('Upload image to ACR') {
   steps{
    script{
     docker.withRegistry("http://${registryURL}", registryCredential){
      dockerImage.push("latest")
      dockerImage.push("${env.BUILD_NUMBER}")
     }
    }
   }
  }
  stage('Deploy on AKS') {
   steps{
```

```
    withKubeConfig([credentialsId: 'AKS-Config', serverUrl: 'https://aks-secdevops-prod-eastus2-dns-7765a616.hcp.eastus2.azmk8s.io:443']) {
      sh 'kubectl apply -f ./k8s-deployment.yaml'
      sh 'kubectl get pods -n ns-backend'
    }
   }
  }
 }
}
```

**Figura 3-29.** Jenkinsfile prueba 1

4. Manifiesto Kubernetes: archivo donde se definen los recursos a desplegar en el clúster de Kubernetes. Para la prueba 1 es necesario desplegar los objetos: Deployment y Service, como se muestra en la Figura 3-30 y Figura 3-31.

k8s-deployment.yaml 862 Bytes

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: backend-java-devsecops
  name: backend-java-devsecops
  namespace: ns-backend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: backend-java-devsecops
  strategy: {}
  template:
    metadata:
      labels:
        app: backend-java-devsecops
    spec:
      containers:
      - name: backend-img
        image: myacrsecdevops.azurecr.io/base-backend-java:latest
        ports:
        - containerPort: 80
        resources:
          requests:
            cpu: 250m
          limits:
            cpu: 500m
      imagePullSecrets:
      - name: acr-secret
```

**Figura 3-30.** Manifiesto. Definición objeto *deployment* usado en la prueba 1

```yaml
apiVersion: v1
kind: Service
metadata:
  name: svc-backend-java-devsecops
  namespace: ns-backend
spec:
  selector:
    app: backend-java-devsecops
  ports:
    - protocol: TCP
      port: 80
  type: LoadBalancer
```

**Figura 3-31.** Manifiesto. Definición objeto *service* usado en la prueba 1

5. Archivo propiedades SonarQube: archivo donde se definen las rutas de los archivos fuente, binarios y librerías tomadas para el análisis de SonarQube, como se muestra en la Figura 3-32.

sonar-project.properties 1.22 KB

Edit | Web IDE | Replace | Delete

```
# Información del proyecto

    sonar.projectVersion=1.0
    sonar.projectDescription= Calidad de Código - SonarQube

    #Datos de los fuentes y binarios
    sonar.sourceEncoding=UTF-8
    sonar.java.source=8
    sonar.sources=restaurante/dominio/src/main/java,restaurante/infraestructura/src/main/java
    sonar.java.binaries=restaurante/dominio/build/classes/java/main,restaurante/infraestructura/build/classes/java/main
    sonar.java.libraries=/var/lib/jenkins/.gradle/caches/modules-2/files-2.1/**/*.jar

    #Datos de los fuentes y binarios de los tests
    sonar.tests=restaurante/dominio/src/test/java,restaurante/infraestructura/src/test/java
    sonar.java.test.binaries=restaurante/dominio/build/classes/java/test,restaurante/infraestructura/build/classes/java/test
    sonar.java.test.libraries=/var/lib/jenkins/.gradle/caches/modules-2/files-2.1/**/*.jar

    #Envío de reportes de JUnit y Jacoco como resultado de la tarea test
    sonar.coverage.jacoco.xmlReportPaths=restaurante/dominio/build/reports/jacoco/test/jacocoTestReport.xml,restaurante/infraestructura/build/reports
    sonar.junit.reportPaths=restaurante/dominio/build/test-results/test,restaurante/infraestructura/build/test-results/test
```

**Figura 3-32.** Archivo usado por SonarQube para el análisis realizado en la prueba 1

Para el correcto funcionamiento del aplicativo, se desplegó una base de datos MySQL en el clúster de Kubernetes a partir del archivo "*deployment-db.yaml*". En este archivo se configuran los objetos de Kubernetes necesarios para el funcionamiento de esta: secret, service y pod. El contenido del archivo construido se muestra en la Figura 3-33.

```
apiVersion: v1
kind: Secret
metadata:
  name: mysql-pass
  namespace: ns-db
type: Opaque
data:
  password: YWRtaW4=

---
apiVersion: v1
kind: Service
metadata:
  name: mysql-service
  namespace: ns-db
  labels:
    name: lbl-k8s-mysql
spec:
  ports:
  - port: 3306
  selector:
    name: lbl-k8s-mysql
  type: ClusterIP

---
```

```
apiVersion: v1
kind: Pod
metadata:
  name: k8s-mysql
  namespace: ns-db
  labels:
    name: lbl-k8s-mysql
spec:
  containers:
  - name: mysql
    image: mysql:latest
    env:
    - name: MYSQL_ROOT_PASSWORD
      valueFrom:
        secretKeyRef:
          name: mysql-pass
          key: password
    ports:
    - name: mysql
      containerPort: 3306
      protocol: TCP
    volumeMounts:
    - name: k8s-mysql-storage
      mountPath: /var/lib/mysql
  volumes:
  - name: k8s-mysql-storage
    emptyDir: {}
```

**Figura 3-33.** Manifiesto de kubernetes para el despliegue de la base de datos MySQL

Se crean los objetos mencionados ejecutando el siguiente comando de kubernetes, como se observa en la Figura 3-34.

```
root@vm-secdevops-prod-eastus2-001:/home/admin_devsecops/manifests/manifest-db# kubectl apply -f deployment-db.yaml
secret/mysql-pass unchanged
service/mysql-service created
pod/k8s-mysql created
root@vm-secdevops-prod-eastus2-001:/home/admin_devsecops/manifests/manifest-db# kubectl get all -n ns-db
NAME            READY   STATUS    RESTARTS   AGE
pod/k8s-mysql   1/1     Running   0          9s

NAME                    TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)    AGE
service/mysql-service   ClusterIP   10.0.124.74   <none>        3306/TCP   9s
```

**Figura 3-34.** Creación de la base de datos MySQL en kubernetes

Luego de la creación del motor base de datos MySQL, es necesario configurar la base de datos *"adn_restaurante"* para el correcto funcionamiento del aplicativo web. Se accede al pod creado con el siguiente comando, como se observa en la Figura 3-35.

```
root@vm-secdevops-prod-eastus2-001:/# kubectl exec k8s-mysql -n ns-db -it -- bash
root@k8s-mysql:/# echo $MYSQL_ROOT_PASSWORD
admin
root@k8s-mysql:/# mysql --user=root --password=$MYSQL_ROOT_PASSWORD
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.27 MySQL Community Server - GPL

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

**Figura 3-35.** Ingreso al pod de la base de datos para su configuración

Se procede a la creación de la base de datos, ver Figura 3-36.

```
mysql> create database adn_restaurante;
Query OK, 1 row affected (0.04 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| adn_restaurante    |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)
```

**Figura 3-36.** Creación base de datos "adn_restaurante"

Adicionalmente a la creación de la base de datos, es necesario configurar la cadena de conexión a la base de datos en el código fuente, como se muestra. Para esto, es importante identificar el DNS de la base de datos MySQL desplegada previamente, Figura 3-37.

```
root@vm-secdevops-prod-eastus2-001:/home/admin_devsecops/manifests/manifest-db# kubectl get svc --all-namespaces
NAMESPACE       NAME                              TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)          AGE
calico-system   calico-kube-controllers-metrics   ClusterIP      10.0.109.254    <none>          9094/TCP         69d
calico-system   calico-typha                      ClusterIP      10.0.183.44     <none>          5473/TCP         69d
default         kubernetes                        ClusterIP      10.0.0.1        <none>          443/TCP          69d
kube-system     kube-dns                          ClusterIP      10.0.0.10       <none>          53/UDP,53/TCP    69d
kube-system     metrics-server                    ClusterIP      10.0.57.231     <none>          443/TCP          69d
ns-backend      svc-backend-java-devsecops        LoadBalancer   10.0.16.223     52.138.111.75   80:30142/TCP     57d
ns-db           mysql-service                     ClusterIP      10.0.124.74     <none>          3306/TCP         4m48s
```

**Figura 3-37.** Visualización objeto service de la base de datos desplegada en kubernetes

Cabe anotar que cuando se habla en términos de kubernetes, el DNS interno se compone de: ***nombre_servicio.namespace.svc.cluster.local:puerto***, por ende, el DNS configurado es: *"mysql-*

*service.ns-db.svc.cluster.local:3306".* Este se añade como una línea de código en el archivo "application.yaml" situado en el código fuente como se puede ver en la Figura 3-38.

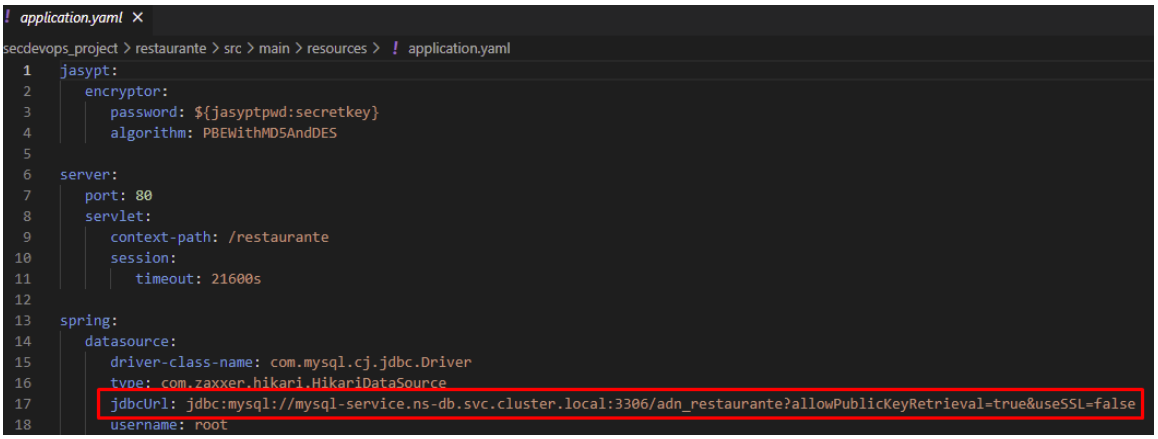



**Figura 3-38**. Configuración cadena conexión a la base de datos

Finalmente, luego de la breve descripción expuesta y el alistamiento de la infraestructura se procede a la ejecución de la prueba 1. Esta inicia con el envío de algún cambio a cualquier archivo localizado en el repositorio mediante un commit a GitLab, que activa el trigger en el "multibranch" configurado en Jenkins y ejecuta el pipeline configurado, ver Figura 3-39.

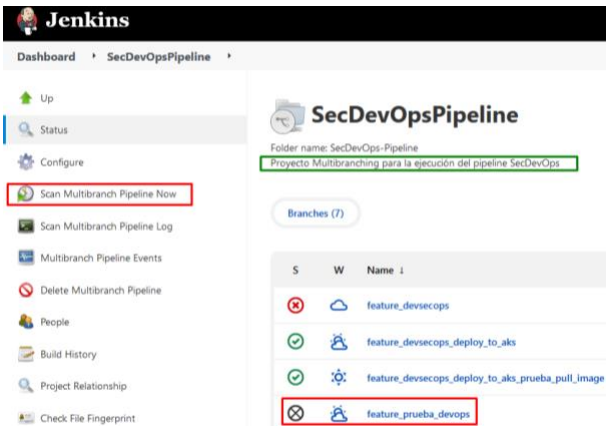



**Figura 3-39.** Activación ejecución pipeline en Jenkins

Automáticamente se ejecuta el pipeline configurado dentro del repositorio y el resultado de la ejecución se observa en la Figura 3-40.


**Figura 3-40.** Ejecución exitosa del pipeline – Prueba 1

Al finalizar el proceso de CI se genera el ejecutable (.JAR) en el repositorio local de Jenkins, situado en la ruta "restaurante/build/libs", como se observa en la Figura 3-41.


**Figura 3-41.** Ejecutable en el repositorio local de Jenkins

Para validar el correcto despliegue del aplicativo web en el clúster de kubernetes se ejecuta el siguiente procedimiento. Primero, se comprueba el estado del pod creado, como se muestra en la Figura 3-42.

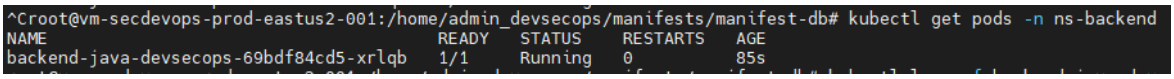



**Figura 3-42.** Visualización estado del pod

Segundo, se procede a la revisión de logs dentro del pod creado para visualizar la inicialización del aplicativo y la conexión a la base de datos, ver en la Figura 3-43.

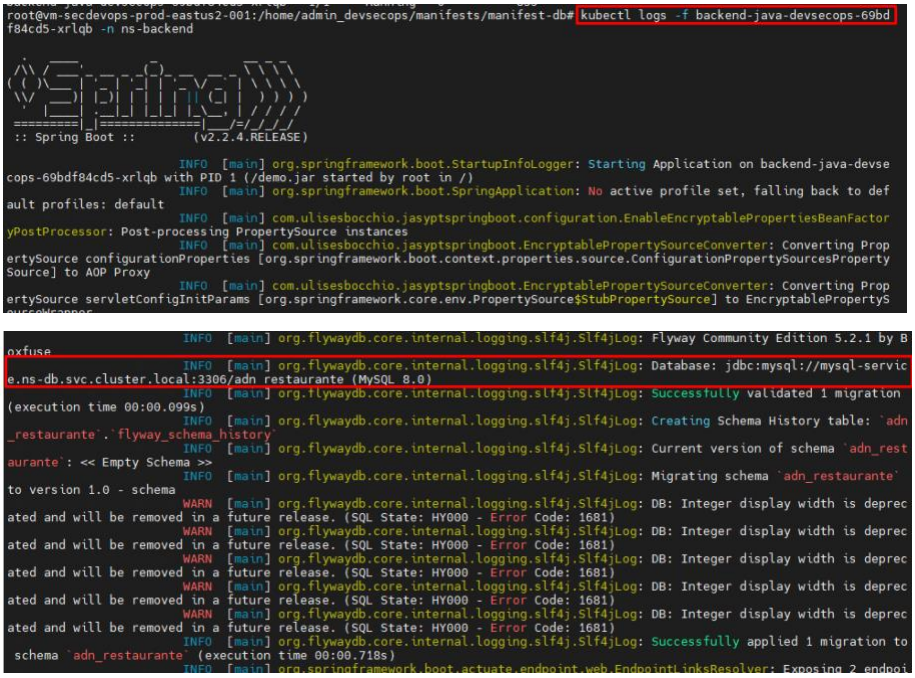



**Figura 3-43.** Inicialización aplicativo web

Y para identificar la dirección IP generada del aplicativo web expuesto, es necesario revisar el servicio creado a partir del manifiesto configurado. La IP pública se puede observar en la Figura 3-44.

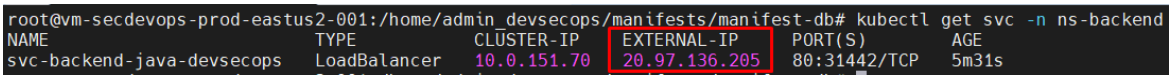



**Figura 3-44.** IP pública para consumir el aplicativo web

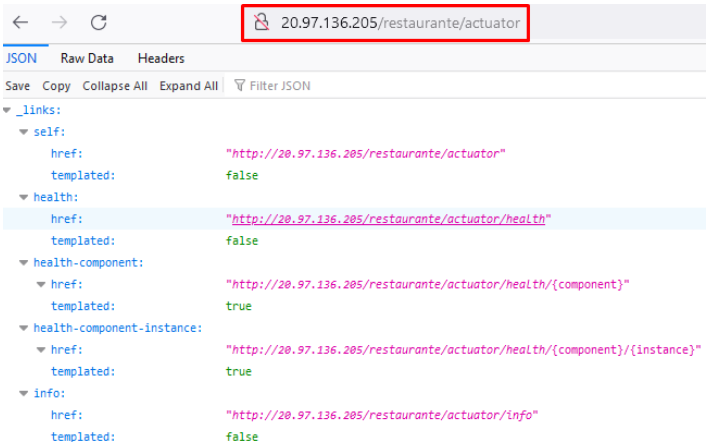



**Figura 3-45.** Despliegue exitoso aplicativo web

Partiendo de la IP pública identificada, se realiza el consumo el aplicativo desde algún navegador web, como se muestra en la Figura 3-45. En este punto, el aplicativo fue desplegado exitosamente empleando la plataforma DevOps configurada.

- ***Prueba 2. Pruebas de seguridad sobre el aplicativo web desplegado sobre la plataforma DevOps***

Partiendo del ejecutable generado por la plataforma DevOps, ver Figura 3-41 y del despliegue exitoso de este expuesto en la Figura 3-45, se presenta el desarrollo de la prueba 2. Para el análisis estático de código y pentesting básico, se seleccionaron cuatro herramientas: AppScan On Cloud, Qualys, Nessus Tenable Essentials y Nikto, diferentes a las contenidas en la Tabla 3-16 y Tabla 3-17; la configuración e instalación de estas herramientas se expone en el *Anexo: Instalación de las herramientas usadas en la prueba dos (2)*. El análisis SAST realizado por la herramienta AppScan On Cloud fue ejecutado mediante el siguiente pipeline configurado en Jenkins.

```
pipeline {
 agent any
 options {
  buildDiscarder(logRotator(numToKeepStr: '3'))
            disableConcurrentBuilds()
 }
 tools {
  jdk 'JDK11'
  gradle 'Gradle5.6'
 }
 stages{
  stage('Checkout') {
   steps{
    echo "------------>Checkout<------------"
                                    checkout scm
   }
  }
  stage('Compile & Unit Tests') {
   steps{
    echo "------------>Compile & Unit Tests<------------"
    sh 'gradle --b ./restaurante/build.gradle clean test'
   }
  }
  stage('Build') {
   steps {
    echo "------------>Build<------------"
                        sh 'gradle --b ./restaurante/build.gradle build -x test'
   }
  }
  stage('AppScan SAST') {
   steps{
    appscan application: '398c029d-43dd-44c0-b1a8-79a4779b503f', credentials: 'api_hcl_appscan', scanner:
static_analyzer(hasOptions: false, target:
'/var/lib/jenkins/workspace/ipeline_feature_prueba_secdevops/restaurante/build/libs/poc-devsecops-0.1.jar'), type: 'Static
Analyzer', name: 'AppScanTestReport - SAST', email: true
   }
  }
 }
```

En la Figura 3-46 se muestra el resultado de la ejecución del pipeline.

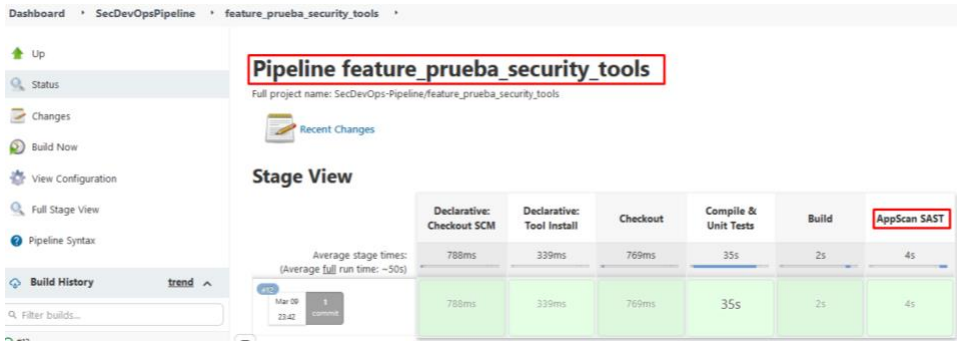



**Figura 3-46.** Ejecución exitosa pipeline para el análisis SAST

Adicionalmente, en la Figura 3-47 se observa la cantidad de hallazgos encontrados clasificados en: 16 altos, 15 medios, 2 bajos, 0 informativos en la ejecución del análisis SAST.

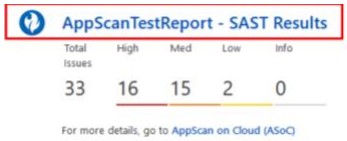


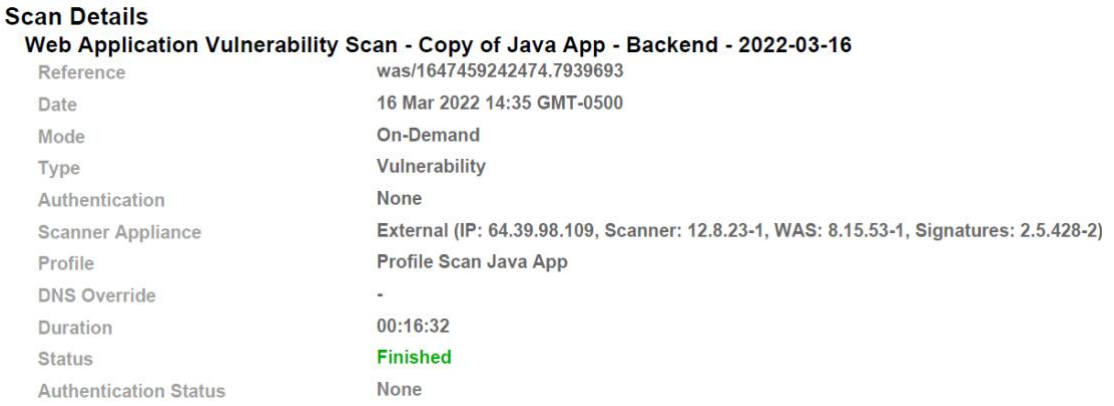

**Figura 3-47.** Resultado SAST – Detalle mínimo

Para la realización del pentesting se muestran en la Tabla 3-29 los endpoints expuestos por el aplicativo web.

**Tabla 3-29.** Endpoints expuestos por el aplicativo web

| ***Nro.*** | ***Endpoints funcionales expuestos por el aplicativo web*** | ***Funcionalidad*** |
|---|---|---|
| 1 | http://20.97.136.205/restaurante/actuator/* | Monitor de SpringBoot |
| 2 | http://20.97.136.205/restaurante/clientes/* | API clientes |
| 3 | http://20.97.136.205/restaurante/platos/* | API platos |
| 4 | http://20.97.136.205/restaurante/usuarios/* | API usuarios |
| 5 | http://20.97.136.205/restaurante/ventas/* | API ventas |
| 6 | http://20.97.136.205/ | IP aplicativo web |
| 7 | http://20.97.136.205/restaurante/ | Endpoint principal |

Fue necesario el acompañamiento del analista de seguridad Julián Valderama Sepúlveda para la ejecución del análisis con la herramienta Qualys. Con esta herramienta se realizó el escaneo de vulnerabilidades de aplicaciones web (Web Application Vulnerability Scan, WAS) a cada una de los endpoints expuestos en la Tabla 3-29, el detalle del escaneo se muestra en la Figura 3-48.



**Scan Details**

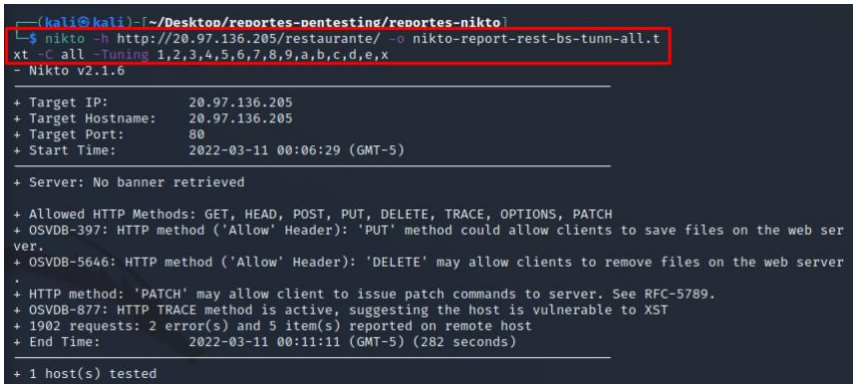
**Web Application Vulnerability Scan - Copy of Java App - Backend - 2022-03-16**

| | |
|---|---|
| Reference | was/1647459242474.7939693 |
| Date | 16 Mar 2022 14:35 GMT-0500 |
| Mode | On-Demand |
| Type | Vulnerability |
| Authentication | None |
| Scanner Appliance | External (IP: 64.39.98.109, Scanner: 12.8.23-1, WAS: 8.15.53-1, Signatures: 2.5.428-2) |
| Profile | Profile Scan Java App |
| DNS Override | - |
| Duration | 00:16:32 |
| Status | Finished |
| Authentication Status | None |

**Figura 3-48.** Detalle escaneo WAS con Qualys – Pentesting

A partir del resultado del análisis expuesto en la Figura 3-48, se obtuvieron en total dieciséis(16) hallazgos clasificados por la herramienta de la siguiente manera: dos (2) vulnerabilidades nivel tres (3) correspondientes a la categoría A2 y A5 de la clasificación de OWASP Top ten 2021, dos (2) hallazgos de contenido sensible correspondiente al nivel A2 y catorce (14) hallazgos de información recolectada asociados a la categoría A5 de OWASP.

En lo que respecta a la herramienta Nikto, en la Figura 3-49 se muestra el comando ejecutado para el análisis realizado al endpoint “http://20.97.136.205/restaurante/”. Del mismo modo que con Qualys, se realiza el análisis a cada uno de los siete endpoints expuestos anteriormente, donde los hallazgos y resultados obtenidos de cada uno se detallan en el siguiente capítulo.



```
┌──(kali㉿kali)-[~/Desktop/reportes-pentesting/reportes-nikto]
└─$ nikto -h http://20.97.136.205/restaurante/ -o nikto-report-rest-bs-tunn-all.t
xt -C all -Tuning 1,2,3,4,5,6,7,8,9,a,b,c,d,e,x
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          20.97.136.205
+ Target Hostname:    20.97.136.205
+ Target Port:        80
+ Start Time:         2022-03-11 00:06:29 (GMT-5)
---------------------------------------------------------------------------
+ Server: No banner retrieved

+ Allowed HTTP Methods: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, PATCH
+ OSVDB-397: HTTP method ('Allow' Header): 'PUT' method could allow clients to save files on the web ser
ver.
+ OSVDB-5646: HTTP method ('Allow' Header): 'DELETE' may allow clients to remove files on the web server
.
+ HTTP method: 'PATCH' may allow client to issue patch commands to server. See RFC-5789.
+ OSVDB-877: HTTP TRACE method is active, suggesting the host is vulnerable to XST
+ 1902 requests: 2 error(s) and 5 item(s) reported on remote host
+ End Time:           2022-03-11 00:11:11 (GMT-5) (282 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

**Figura 3-49.** Ejecución exitosa del escaneo con Nikto – Pentesting

Sobre los endpoints expuestos en la Tabla 3-29 se obtuvieron nueve (9) hallazgos clasificados como sigue a continuación. Para el endpoint nro (6) se obtuvieron tres (3) hallazgos y para los otros seis restantes (6) endpoints se obtuvieron el total de seis (6) hallazgos clasificados con respecto a la base de datos de vulnerabilidades OSVBD como: OSVBD-397, OSVBD-5646 y OSVBD-877.

En la Figura 3-50 se observan los diferentes tipos de análisis que ofrece la herramienta Nessus. En nuestro caso, el tipo de análisis seleccionado fue: “Advanced Scan”.

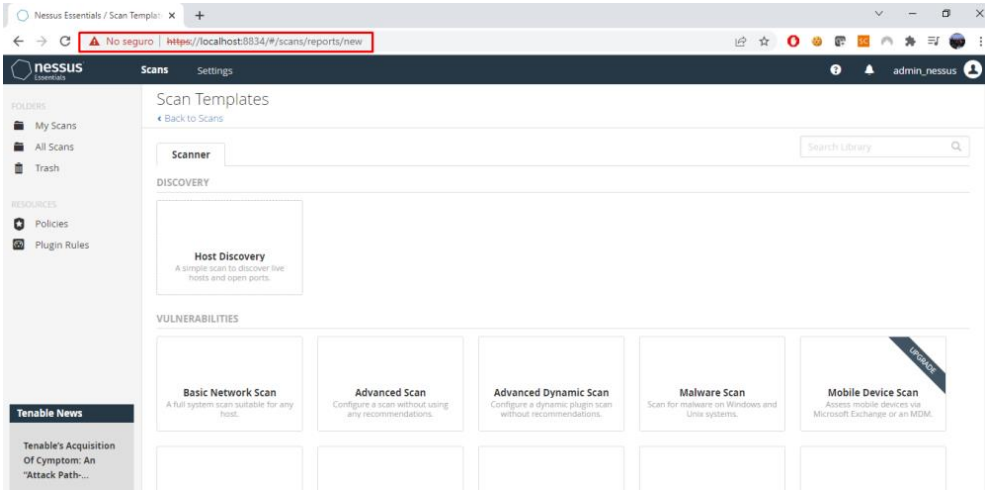



**Figura 3-50.** Tipos de análisis ofrecidos por Nessus Essentials

Y finalizando con los hallazgos obtenidos con la herramienta Nessus, el análisis fue realizado sobre el endpoint http://20.97.136.205 arrojando cuatro (4) hallazgos clasificados como severidad informativa.

- ***Prueba 3. Aplicativo web ejecutado y desplegado sobre la plataforma SecDevOps***

En base a la plataforma DevOps desplegada en la prueba 1 se construye la plataforma SecDevOps con las herramientas de seguridad propuestas en la estrategia de seguridad establecida. Tomando en cuenta esto, se expone el pipeline tanto para CI como para CD, indicando los puntos donde se generan los hallazgos de cada paso de seguridad y los 2 puntos establecidos para romper el build (RB). Así como se expuso conceptualmente en la estrategia de seguridad propuesta, en cada punto se realiza la evaluación de los hallazgos consolidados para determinar si hay una vulnerabilidad o un falso positivo; de acuerdo con esto, las vulnerabilidades se analizan según la política establecida en la propuesta de seguridad y se toma la decisión de si se debe romper o no el build (RB).

```
pipeline {
  agent any
  environment {
    url_target= "http://20.97.136.205"
    registryCredential= "ACR"
    registryURL = "myacrsecdevops.azurecr.io"
    registryName = "base-backend-java"
    dockerImage= ""
  }
  options {
    buildDiscarder(logRotator(numToKeepStr: '3'))
```

```
            disableConcurrentBuilds()
 }
 tools {
  jdk 'JDK11'
  gradle 'Gradle5.6'
 }
 stages{
  stage('Checkout') {
   steps{
    echo "------------>Checkout SCM<------------"
                                        checkout scm
   }
  }
//CONTINUOUS INTEGRATION
  stage('Compile & Unit Tests') {
   steps{
    echo "------------>Compile & Unit Tests<------------"
    sh 'gradle --b ./restaurante/build.gradle clean test'
   }
  }
  stage('Build') {
   steps {
    echo "------------>Build<------------"
                            sh 'gradle --b ./restaurante/build.gradle build -x test'
   }
  }
  stage('Dependency Analysis - SCA') {
   steps {
    echo "----->Dependency Analysis - SCA<-------"
     dependencyCheck additionalArguments: '''
      --out "./"
      --scan "./restaurante/build/libs/*.jar"
      --format "XML"
      --log "./log-AD.log"
      --prettyPrint ''', odcInstallation: 'Dependency-Check'
      dependencyCheckPublisher pattern: 'dependency-check-report.xml'
   }
  }
// Se genera el reporte con los hallazgos de: Análisis de dependencias/Análisis composición de código.
  stage('Static Code Analysis') {
   steps{
      echo '------------>SAST Analysis<------------'
      withSonarQubeEnv('SonarQube') {
       sh "${tool name: 'sonar_scanner2', type:'hudson.plugins.sonar.SonarRunnerInstallation'}/bin/sonar-scanner -
Dsonar.projectKey=co.com.tesis.devsecops.poc.${BRANCH_NAME} -Dsonar.projectName=Tesis-DevSecOpsPoC.${BRANCH_NAME} -
Dproject.settings=sonar-project.properties"
      }
   }
  }
// Se genera el reporte con los hallazgos de: Análisis de código estático - SAST
  stage('Build Docker Image') {
   steps{
    echo "------------>Build Docker Image<------------"
    script{
     dockerImage = docker.build registryName
    }
   }
```

## //EVALUACIÓN DEL PARÁMETRO RB

```
//CONTINUOUS DEPLOYMENT
  stage('Container Security Scan') {
   steps {
```

```
      echo "------------>Anchore Scannig<-------------"
      sh 'echo "${registryURL}/${registryName}:latest `pwd`/Dockerfile" > anchore_images'
      anchore name: 'anchore_images', bailOnFail: "false", forceAnalyze: "true"
     }
    }
// Se genera el reporte con los hallazgos de: Análisis de seguridad en contenedores
    stage('Upload Image to ACR') {
     steps{
      echo "------------>Upload Image to ACR<------------"
      script{
       docker.withRegistry("http://${registryURL}", registryCredential){
        dockerImage.push("latest")
        dockerImage.push("${env.BUILD_NUMBER}")
       }
      }
     }
    }
    stage('Deploy on AKS') {
     steps{
      echo "------------>Deploy on AKS<------------"
      withKubeConfig([credentialsId: 'AKS-Config', serverUrl: 'https://aks-secdevops-prod-eastus2-dns-7765a616.hcp.eastus2.azmk8s.io:443']) {
       sh 'kubectl apply -f ./k8s-deployment.yaml'
       sh 'kubectl get pods -n ns-backend'
      }
     }
    }
    stage('Dynamic Code Scanning') {
     steps {
      echo "------------>DAST Analysis<------------"
      script {
       def data = "sudo docker stop owasp2\nsudo docker rm owasp2\nsudo docker pull owasp/zap2docker-stable\nsudo docker run -dt --name owasp2 owasp/zap2docker-stable /bin/bash\nsudo docker exec owasp2 mkdir /zap/wrk\nsudo docker exec owasp2 zap-full-scan.py -t ${url_target} -r report-full-scan-DAST.html -a -d -j -I\nsudo docker cp owasp2:/zap/wrk/report-full-scan-DAST.html /home/admin_devsecops/dast-report/report-full-scan-DAST.html\n"
       def remote = [:]
       remote.name = 'Zap-Server'
       remote.host = '192.16.2.4'
       remote.user = 'admin_devsecops'
       remote.password = '@@Admin_DevSecOps**'
       remote.allowAnyHosts = true
       writeFile file: 'baseline-scan-zap.sh', text: data
       sshScript remote: remote, script: "baseline-scan-zap.sh"
      }
     }
// Se genera el reporte con los hallazgos de: Análisis dinámico de seguridad - DAST
    }
   }
}
```

## //EVALUACIÓN DEL PARÁMETRO RB

Luego de la ejecución del pipeline SecDevOps, se observa en la Figura 3-51 el resultado de este.

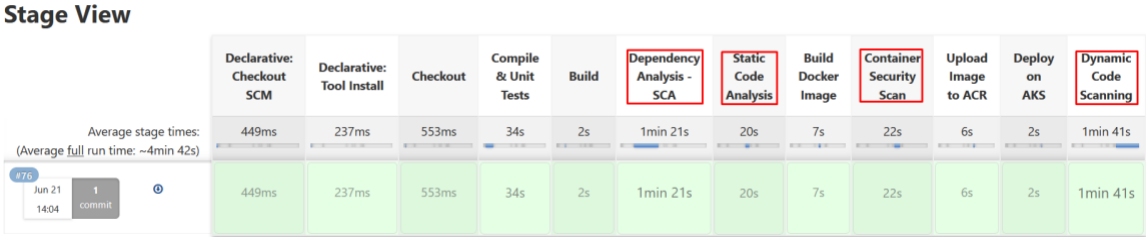


**Figura 3-51.** Resultado ejecución pipeline SecDevOps

Adicionalmente, se muestra el detalle obtenido en Jenkins de los pasos de seguridad: análisis dependencias-SCA, análisis estático de código y análisis en imágenes Docker. Este detalle se muestra a continuación en las Figura 3-52 y Figura 3-53.

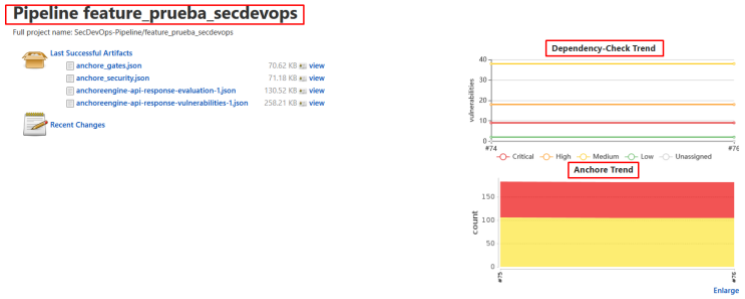



**Figura 3-52.** Detalle resultado pipeline SecDevOps - 1

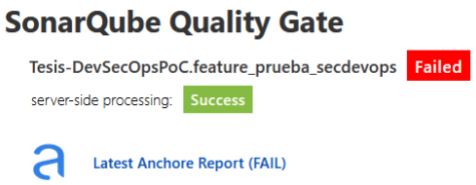



**Figura 3-53.** Detalle resultado pipeline SecDevOps – 2

El análisis de seguridad en kubernetes se realiza bajo el siguiente comando expuesto en la Figura 3-54.

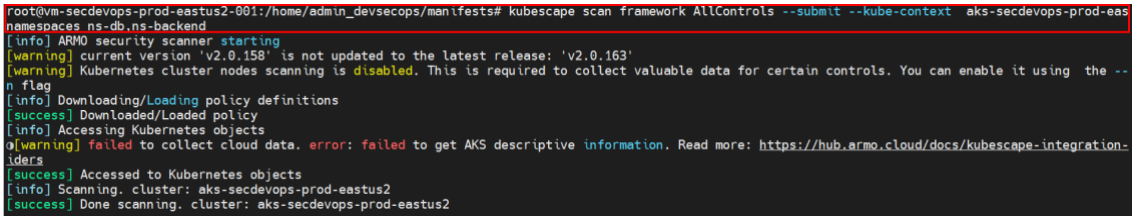



**Figura 3-54.** Detalle resultado pipeline SecDevOps – 2

También, en la Figura 3-55, se visualiza el contenido del repositorio local de Jenkins luego de la ejecución del pipeline de SecDevOps.

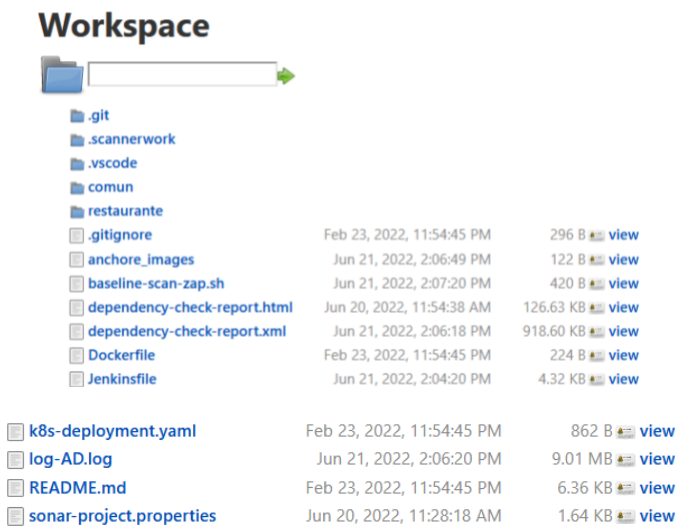



**Figura 3-55.** Repositorio generado luego de la ejecución del pipeline SecDevOps

El escaneo DAST es ejecutado y almacenado en la máquina donde se está desplegado GitLab, en la Figura 3-56 se puede visualizar el reporte generado en ".html".

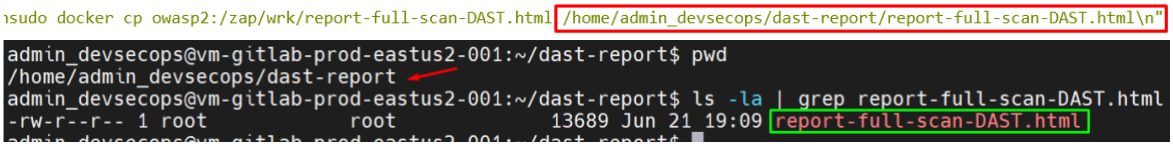



**Figura 3-56.** Reporte generado por DAST

## 3.4.3. Hallazgos encontrados y resultados obtenidos

- ***Hallazgos Prueba 1***

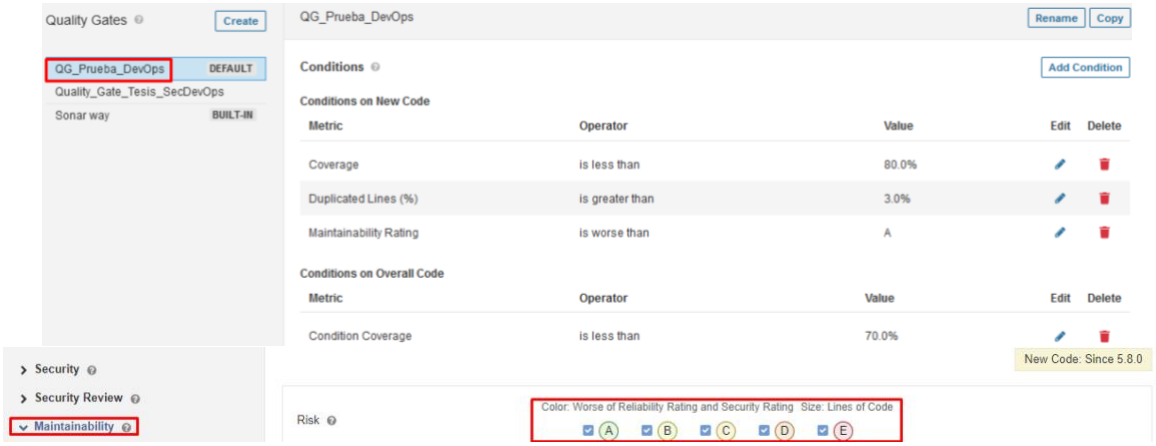



**Figura 3-57.** Quality Gate "QG" configurado – Clasificación mantenibilidad

A partir de la ejecución del pipeline y el despliegue del aplicativo web evidenciados en la Figura 3-41 y Figura 3-45, respectivamente, a continuación se mencionan los hallazgos y resultados obtenidos. Para la ejecución de la prueba 1 se configuró el Quality Gate (QG) "*QG_Prueba_DevOps*" en SonarQube cumpliendo con el objetivo de la metodología DevOps, el cual se enfoca en el análisis de la calidad de código, cobertura y mantenibilidad del código, como se puede ver en la Figura 3-57. Cabe mencionar que, las medidas en la herramienta SonarQube se realizan sobre una calificación "rating" entre A y E para su estimación, dada esta explicación, el resultado de la evaluación del QG se expone en la Figura 3-58.

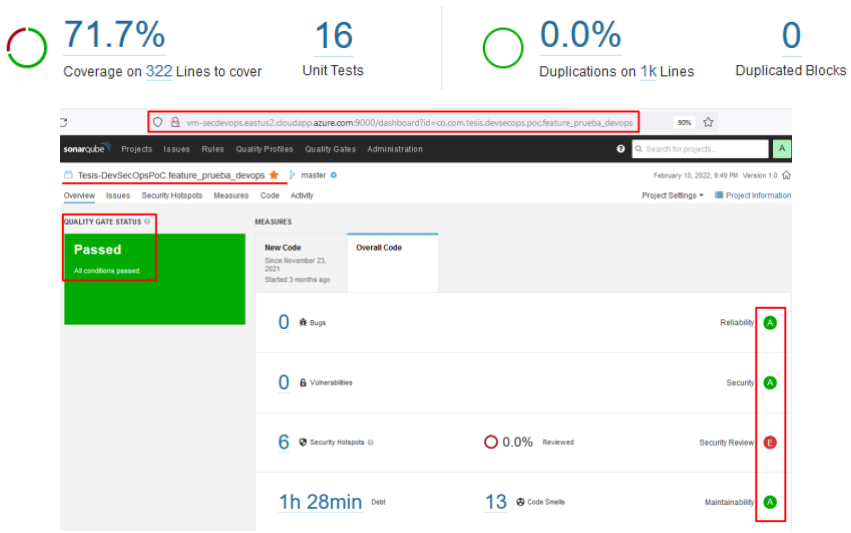



**Figura 3-58.** Resultado de la evaluación del Quality Gate – Cobertura/Mantenibilidad

De acuerdo al resultado del análisis SAST sobre el código expuesto en la Figura 3-58, se concluye que los valores mencionados se encuentran dentro del rango óptimo y permitido por el QG. Dado esto, en la Figura 3-59 se expone el estado del QG evaluado.

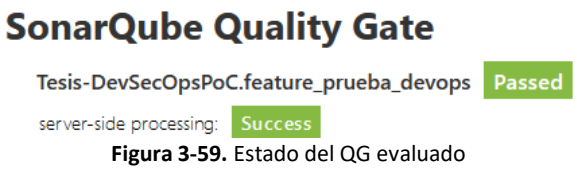



**Figura 3-59.** Estado del QG evaluado

Partiendo de la definición conceptual de DevOps abordada en el capítulo 1.1.3, se verifica que esta metodología se enfoque en adoptar la automatización desde las fases iniciales del desarrollo de software y el despliegue de este y no en incluir la seguridad a lo largo de la metodología, por lo cual es probable que el aplicativo web desplegado incluya brechas de seguridad que pueden ser

explotadas y ejecutadas como vulnerabilidades. Adicionalmente, esta es la forma tradicional de operación con DevOps, donde la seguridad es incluida después del despliegue del aplicativo mediante pruebas y análisis de seguridad, lo que puede conllevar a un riesgo y reproceso.

- ***Hallazgos Prueba 2***

Con base en el resultado del pipeline ejecutado en la prueba 1, a continuación, en la Figura 3-60 se observan los puntos donde se realizaron los análisis propuestos: SAST, pentesting y las herramientas seleccionadas para la ejecución de la prueba 2.

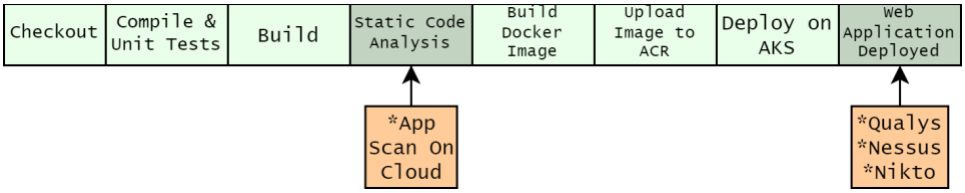



**Figura 3-60.** Puntos análisis Prueba 2

Se presentan los resultados del análisis de código estático (SAST) realizados al ejecutable "poc-devsecops-0.1.jar" generado en el pipeline. Se obtuvieron (33) hallazgos: 16 altos, 15 medios y 2 bajos, cada uno de estos clasificados conforme al listado de los 10 riesgos más críticos en las aplicaciones web. En la Figura 3-61, se muestra el top 10 OWASP de los riesgos más críticos del año 2021 en contraste con el top 10 del año 2017.

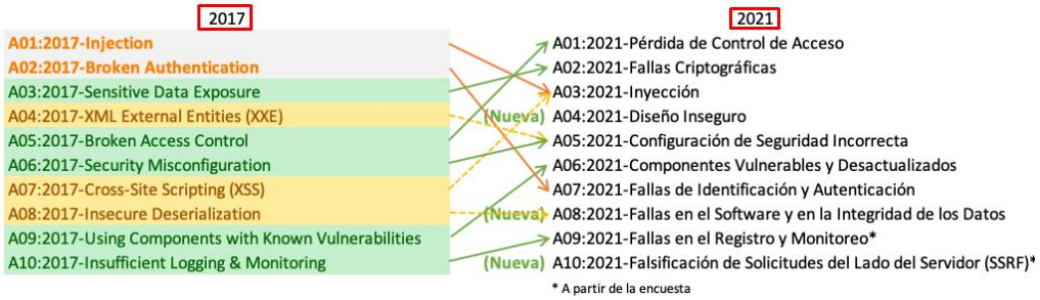



**Figura 3-61.** OWASP Top 10: 2021. Comparación con el Top10:2017 **[30]**

Los 33 hallazgos encontrados corresponden a la categoría A082021: fallas en el software y en la integración de los datos del OWASP Top10: 2021.

| H | A8 - Software and Data Integrity Failures 33 |
|---|---|
| SAST - | Open Source Component |
| Risk: | It is possible for an attacker to use the web server to attack other sites, which increases his or her anonymity |
| Cause: | Latest patches or hotfixes for 3rd. party products were not installed |
| Threat Classification: | Abuse of Functionality |
| CWE: | 829 |

**Figura 3-62.** Detalle hallazgos encontrados en el análisis SAST

En la Figura 3-62 se muestra el detalle del tipo SAST, el riesgo al que está expuesto, la causa o el porqué del riesgo, la clasificación de la amenaza según la herramienta y la CWE asociada de los hallazgos encontrados.

En la Figura 3-63, se visualizan los hallazgos calificados con severidad alta.

| Severity | Location |
|---|---|
| High | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/log4j-api-2.12.1.jar |
| High | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/log4j-api-2.12.1.jar |
| High | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/log4j-api-2.12.1.jar |
| High | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/logback-core-1.2.3.jar |
| High | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/protobuf-java-3.11.4.jar |
| High | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/jackson-databind-2.10.2.jar |
| High | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/protobuf-java-3.11.4.jar |
| High | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/snakeyaml-1.25.jar |
| High | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/spring-core-5.2.3.RELEASE.jar |
| High | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/undertow-core-2.0.29.Final.jar |
| High | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/undertow-core-2.0.29.Final.jar |
| High | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/undertow-core-2.0.29.Final.jar |
| High | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/undertow-servlet-2.0.29.Final.jar |
| High | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/undertow-servlet-2.0.29.Final.jar |
| High | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/undertow-websockets-jsr-2.0.29.Final.jar |
| High | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/undertow-websockets-jsr-2.0.29.Final.jar |

**Figura 3-63.** Hallazgos severidad alta

Dado el resultado anterior, nos centramos en el análisis del hallazgo de severidad alta: “log4j-api-2.12.1.jar”, identificado CVE: 2021-44228 y CWE: 829. Este hallazgo es identificado como vulnerabilidad llamada “Log4Shell, de ejecución remota de código (RCE) que permite a los agentes maliciosos ejecutar código Java arbitrario, tomando el control de un servidor de destino” [125]. Vulnerabilidad ejecutada en el año 2021 calificada en la CVSS con 10 puntos. Su fallo fue encontrado en la librería de Java Log4j que es usada masivamente en sistemas empresariales y aplicaciones web. Evidentemente, fue una de las vulnerabilidades más alertadas por diferentes agencias de ciberseguridad a nivel mundial para su conocimiento y prevención. Una de las recomendaciones para la solución es actualizar la versión a partir de 2.15.0 [126] [127].

Todos los detalles de la vulnerabilidad Log4Shell encontrada por la herramienta AppScan On Cloud se pueden visualizar en la Figura **3-64**.

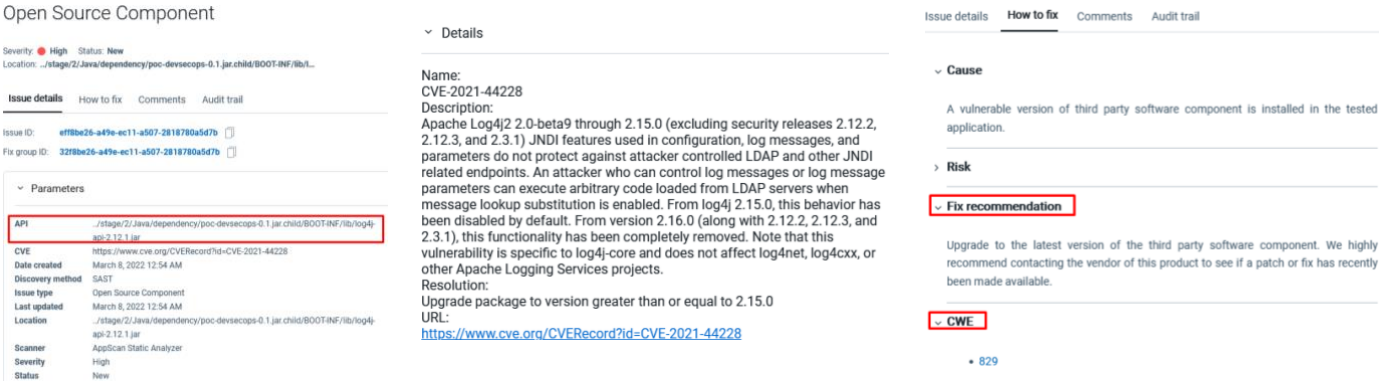



**Figura 3-64**. Detalles vulnerabilidad Log4Shell

Hallazgos calificados con severidad media, Figura 3-65.

| | |
|---|---|
| Medium | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/jakarta.el-3.0.3.jar |
| Medium | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/commons-io-2.6.jar |
| Medium | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/hibernate-validator-6.0.18.Final.jar |
| Medium | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/guava-20.0.jar |
| Medium | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/log4j-api-2.12.1.jar |
| Medium | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/junit-4.12.jar |
| Medium | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/undertow-core-2.0.29.Final.jar |
| Medium | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/undertow-core-2.0.29.Final.jar |
| Medium | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/undertow-core-2.0.29.Final.jar |
| Medium | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/undertow-servlet-2.0.29.Final.jar |
| Medium | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/undertow-servlet-2.0.29.Final.jar |
| Medium | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/undertow-servlet-2.0.29.Final.jar |
| Medium | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/undertow-websockets-jsr-2.0.29.Final.jar |
| Medium | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/undertow-websockets-jsr-2.0.29.Final.jar |
| Medium | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/undertow-websockets-jsr-2.0.29.Final.jar |

**Figura 3-65.** Hallazgos severidad media

Hallazgos calificados con severidad baja, Figura 3-66.

| | |
|---|---|
| Low | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/log4j-api-2.12.1.jar |
| Low | ../stage/2/Java/dependency/poc-devsecops-0.1.jar.child/BOOT-INF/lib/guava-20.0.jar |

**Figura 3-66.** Hallazgos severidad baja

Para el análisis del aplicativo web realizado con la herramienta Nikto, se tomaron como base los endpoints expuestos en la Tabla 3-29. El resultado de dicho análisis realizado a los endpoints 5, 6 y 7 se muestra en: Figura 3-67, Figura 3-68 y Figura 3-69.

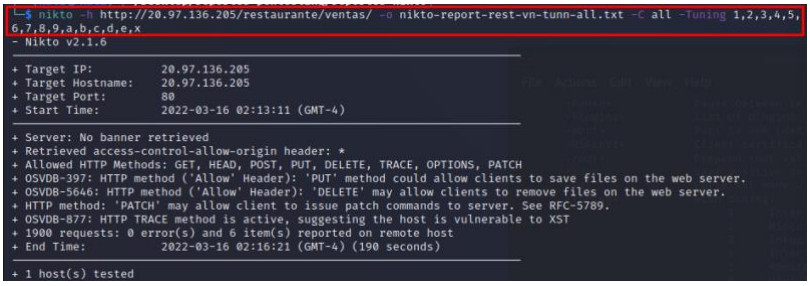



**Figura 3-67.** Resultado análisis a: http://20.97.136.205/restaurante/ventas/

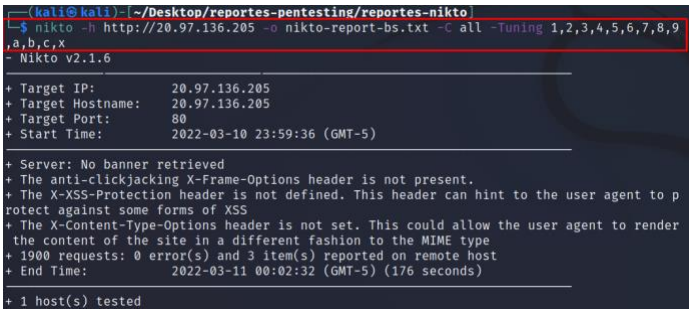



**Figura 3-68.** Resultado análisis a: http://20.97.136.205/

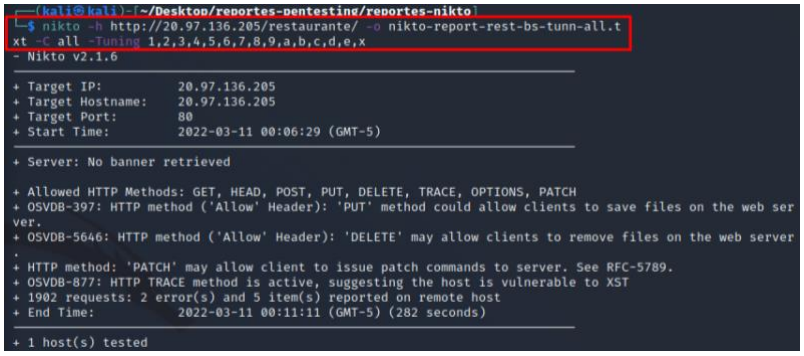



**Figura 3-69.** Resultado análisis a: http://20.97.136.205/restaurante/

Cabe aclarar que para ciertos endpoints los resultados son totalmente iguales, por ello, la consolidación y el detalle de todos los hallazgos que se refieren a los endpoints, se exponen en la Tabla 3-30.

**Tabla 3-30.** Resultados análisis con la herramienta Nikto

| ***Endpoint analizado*** | ***Hallazgos*** | ***Detalle del reporte de Nikto*** |
|---|---|---|
| *http://20.97.136.205/* | GET The anti-clickjacking | GET The anti-clickjacking X-Frame-Options header is not present. |
| | GET The X-XSS-Protection header is not defined | This header can hint to the user agent to protect against some forms of XSS |
| | GET the X-Content-Type-Options header is not set | This could allow the user agent to render the content of the site in a different fashion to the MIME type |
| | OPTIONS Allowed HTTP Methods: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, PATCH | |

| | | |
|---|---|---|
| *http://20.97.136.205/restaurante/clientes/*<br>*http://20.97.136.205/restaurante/platos/*<br>*http://20.97.136.205/restaurante/ventas/* | OSVDB-397 | HTTP method ('Allow' Header): 'PUT' method could allow clients to save files on the web server. |
| | OSVDB-5646 | HTTP method ('Allow' Header): 'DELETE' may allow clients to remove files on the web server. |
| | HTTP method: 'PATCH' | May allow client to issue patch commands to server. See RFC-5789. |
| | OSVDB-877 | TRACE HTTP TRACE method is active, suggesting the host is vulnerable to XST |
| | Retrieved access-control-allow-origin header: * | CORS |
| *http://20.97.136.205/restaurante/usuarios/*<br>*http://20.97.136.205/restaurante/actuator/*<br>*http://20.97.136.205/restaurante/* | OPTIONS Allowed HTTP Methods: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, PATCH | |
| | OSVDB-397 | HTTP method ('Allow' Header): 'PUT' method could allow clients to save files on the web server. |
| | OSVDB-5646 | HTTP method ('Allow' Header): 'DELETE' may allow clients to remove files on the web server. |
| | HTTP method: 'PATCH' | May allow client to issue patch commands to server. See RFC-5789. |
| | OSVDB-877 | TRACE HTTP TRACE method is active, suggesting the host is vulnerable to XST |

El resultado obtenido por el análisis realizado con la herramienta Qualys se consolida en la Tabla 3-31, la cual se expone a continuación.

**Tabla 3-31.** Resultados análisis con la herramienta Qualys

| ***Cantidad hallazgos*** | ***Endpoints*** | ***Resultados*** | ***Grupo*** | ***Detalles escaneo WAS*** | ***Severidad*** | ***Clasificación OWASP (2021)*** |
|---|---|---|---|---|---|---|
| 2 | *http://20.97.136.205/*<br>*http://20.97.136.205/restaurante/** | Vulnerability (Level 3)<br>**Security Risk: Medium** | Information Disclosure | 150079 Slow HTTP headers vulnerability | Potential Vulnerability - Level 3 | A5 Security Misconfiguration |
| | | | | 150263 Insecure Transport | Confirmed Vulnerability - Level 3 | A2 Cryptographic Failures |
| 2 | *http://20.97.136.205/restaurante/** | Sensitive Contents | Custom Contents | 150016 Sensitive Content In HTML | Sensitive Content – Level 2 | A2 Cryptographic Failures |
| 12 | *http://20.97.136.205/*<br>*http://20.97.136.205/restaurante/** | Information Gathered | Security Weaknesses (5) | 150202 Missing header: X-Content-Type-Options | Information Gathered - Level 2 | A5 Security Misconfiguration |
| | | | | 150206 Content-Security-Policy Not Implemented | | |
| | | | | 150208 Missing header: Referrer-Policy | | |
| | | | | 150262 Missing header: Feature-Policy | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | 150204 Missing header: X-XSS-Protection | Severity Information Gathered - Level 1 | |
| | | | Scan Diagnostics (7) | | | |

Finalmente, el resultado obtenido con herramienta Nessus se puede apreciar en la Tabla 3-32.

**Tabla 3-32.** Resultados análisis con la herramienta Nessus

| ***Endpoint analizado*** | ***Resultados*** | ***Grupo*** | ***Reporte Nessus*** | ***Severidad*** |
|---|---|---|---|---|
| http://20.97.136.205 | HTTP Methods Allowed (per directory) | WebServers | The following HTTP methods are considered insecure: PUT, DELETE, CONNECT, TRACE, HEAD | INFO |
| | HyperText Transfer Protocol (HTTP) Information | WebServers | This test gives some information about the remote HTTP protocol - the version used, whether HTTP Keep-Alive and HTTP pipelining are enabled | |
| | Nessus Scan Information | Settings | Information | |
| | Nessus SYN scanner | Port scanners | This plugin is a SYN 'half-open' port scanner. It shall be reasonably quick even against a firewalled target | |
| | Ping the remote host | Port scanners | Nessus was able to determine if the remote host is alive | |

Consolidando el resultado del análisis de cada una de las herramientas empleadas se puede mencionar qué: Hay ausencia de configuración en cabeceras que pueden proveer seguridad en el aplicativo, se permite el consumo más métodos HTTP a los necesarios lo que puede concluir en la obtención de información sensible y, por último, el aplicativo web es desplegado bajo el protocolo no seguro HTTP, permitiendo la posible interceptación de información no cifrada.

En consecuencia, al resultado consolidado de cada una de las herramientas empleadas se puede mencionar que hay ausencia de configuración en cabeceras que pueden proveer seguridad en el aplicativo; se permite el consumo de más métodos HTTP de los necesarios, lo que puede concluir en la obtención de información sensible y, finalmente, el aplicativo web es desplegado bajo el protocolo no seguro HTTP, permitiendo la posible interceptación de información no cifrada.

El análisis de código estático SAST y el pentesting son las pruebas de seguridad más conocidas y ejecutadas en el campo de la seguridad. Aclarando que las compañías contratan estos servicios con terceros, ya que el equipo conformado no cuenta con la capacidad suficiente o con los conocimientos necesarios para ejecutarlos. Los resultados de cada análisis con sus respectivos controles para remediar las brechas de seguridad se presentan a la compañía, esta toma la decisión de si los implementa o, por el contrario, la empresa contratada, además de ejecutar los análisis, debe implementarlos. Se evidencia que en la gran mayoría de las ocasiones no existe automatización en estos análisis, por lo que puede pasar una gran cantidad de tiempo para conocer el resultado con el estado de seguridad de la aplicación y, generalmente, cuando estos resultados son socializados al departamento encargado, la aplicación se encuentra en producción con brechas de seguridad.

- ***Hallazgos Prueba 3***

Tomando como referencia la estrategia de seguridad propuesta en la cual se definió el dónde y el cómo romper el build, se muestran los resultados obtenidos de la ejecución de la prueba 3. De conformidad con el resultado del pipeline SecDevOps, se muestran a continuación los hallazgos encontrados a partir de la evaluación de seguridad en: análisis de dependencias, análisis de composición de código y análisis de código estático.

En el análisis de dependencias y composición de código se obtuvieron 67 hallazgos clasificados en: 9 críticos, 18 altos, 38 medios, 2 bajos. El detalle de los hallazgos se puede visualizar en la Figura 3-70 y Figura 3-71.

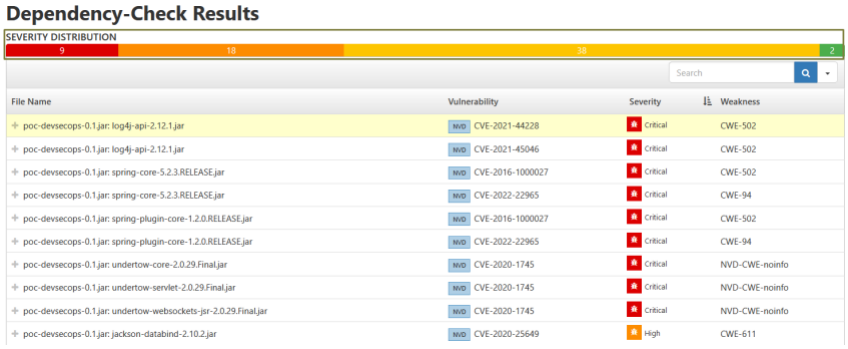



**Figura 3-70.** Resultado análisis de dependencias y composición de código

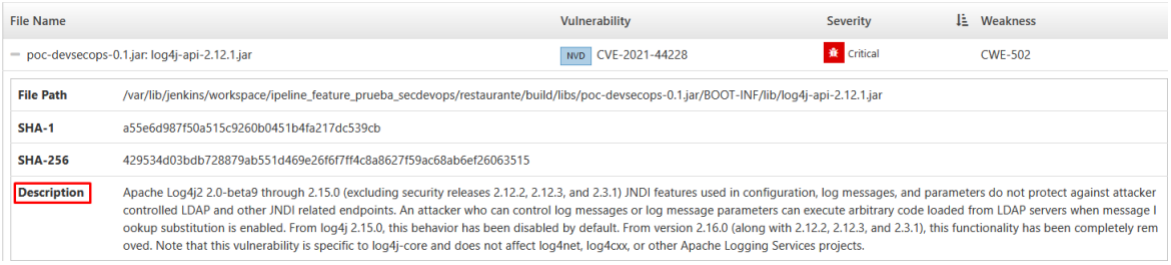



**Figura 3-71**. Resultado análisis de dependencias y composición de código - Detalle

Por parte del análisis estático de código se obtuvieron 6 hallazgos clasificados como security hotspots desde la herramienta SonarQube con severidad baja. El resultado se puede observar en la Figura 3-72.

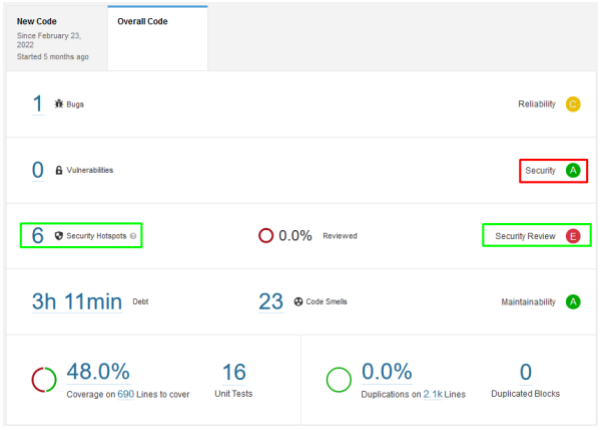



**Figura 3-72.** Resultado análisis estático de código SAST

En la Figura 3-73 se muestra mayor detalle de los 6 security hotspots relacionados a CORS.

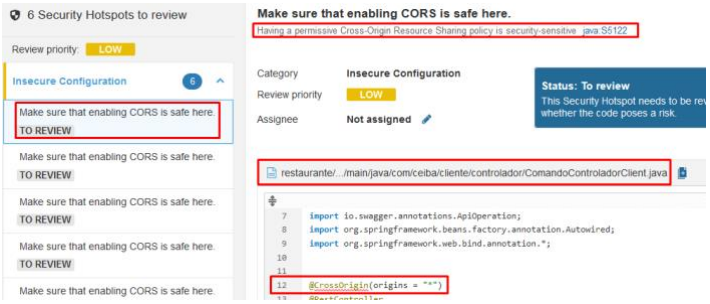



**Figura 3-73.** Resultado análisis estático de código SAST –Detalle Security Hotspots

Desde la herramienta SonarQube también se puede visualizar la CVE, el riesgo al que está expuesta la aplicación web y la posible solución. En la Figura 3-74 se observa el detalle mencionado previamente.

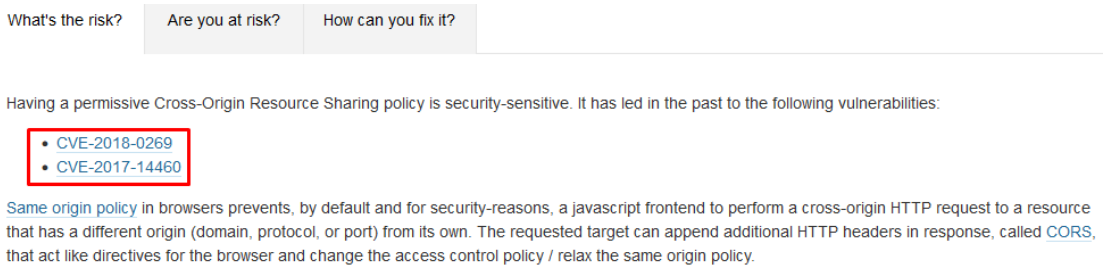



**Figura 3-74.** Detalle para el hallazgo encontrado

En este punto, luego de obtener los hallazgos del análisis de dependencias, análisis de composición de código y análisis de código estático el analista de seguridad dispone del reporte consolidado de todos los hallazgos obtenidos previamente y procede a confirmar si corresponden a una vulnerabilidad o un falso positivo. Para facilidad del ejercicio, en el consolidado de los hallazgos solo se visualizan los calificados con severidad crítica y alta, como se puede visualizar en la Tabla 3-33 y Tabla 3-34.

**Tabla 3-33.** Hallazgos en el CI de: Análisis dependencias y composición de código –Prueba 3

| *Nro.* | *Severidad* | *Nombre archivo* | *Componente* | *CVE asociada* | *(Debilidad) - CWE asociada* |
|---|---|---|---|---|---|
| 1 | Críticas | poc-devsecops-0.1.jar: log4j-api-2.12.1.jar | NVD | CVE-2021-44228 | CWE-502 |
| 2 | | poc-devsecops-0.1.jar: log4j-api-2.12.1.jar | NVD | CVE-2021-45046 | CWE-502 |
| 3 | | poc-devsecops-0.1.jar: spring-core-5.2.3.RELEASE.jar | NVD | CVE-2016-1000027 | CWE-502 |
| 4 | | poc-devsecops-0.1.jar: spring-core-5.2.3.RELEASE.jar | NVD | CVE-2022-22965 | CWE-94 |
| 5 | | poc-devsecops-0.1.jar: spring-plugin-core-1.2.0.RELEASE.jar | NVD | CVE-2016-1000027 | CWE-502 |
| 6 | | poc-devsecops-0.1.jar: spring-plugin-core-1.2.0.RELEASE.jar | NVD | CVE-2022-22965 | CWE-94 |
| 7 | | poc-devsecops-0.1.jar: undertow-core-2.0.29.Final.jar | NVD | CVE-2020-1745 | NVD-CWE-noinfo |
| 8 | | poc-devsecops-0.1.jar: undertow-servlet-2.0.29.Final.jar | NVD | CVE-2020-1745 | NVD-CWE-noinfo |
| 9 | | poc-devsecops-0.1.jar: undertow-websockets-jsr-2.0.29.Final.jar | NVD | CVE-2020-1745 | NVD-CWE-noinfo |
| 10 | | poc-devsecops-0.1.jar: jackson-databind-2.10.2.jar | NVD | CVE-2020-25649 | CWE-611 |
| 11 | Alta | poc-devsecops-0.1.jar: jackson-databind-2.10.2.jar | NVD | CVE-2020-36518 | CWE-787 |
| 12 | | poc-devsecops-0.1.jar: snakeyaml-1.25.jar | NVD | CVE-2017-18640 | CWE-776 |
| 13 | | poc-devsecops-0.1.jar: spring-boot-2.2.4.RELEASE.jar | NVD | CVE-2022-27772 | CWE-668 |
| 14 | | poc-devsecops-0.1.jar: spring-boot-actuator-2.1.6.RELEASE.jar | NVD | CVE-2022-27772 | CWE-668 |
| 15 | | poc-devsecops-0.1.jar: spring-core-5.2.3.RELEASE.jar | NVD | CVE-2021-22118 | CWE-269 |
| 16 | | poc-devsecops-0.1.jar: undertow-core-2.0.29.Final.jar | OSSINDEX | CVE-2019-14888 | CWE-400 |

| | | | | | |
|---|---|---|---|---|---|
| 17 | | poc-devsecops-0.1.jar: undertow-core-2.0.29.Final.jar | NVD | CVE-2020-10705 | CWE-770 |

**Tabla 3-34.** Hallazgos en el CI de: Análisis estático de código –Prueba 3

| *Nro.* | *Severidad* | *Nombre archivo* | *CVE asociada* |
|---|---|---|---|
| 1 | Baja | main/java/com/ceiba/cliente/controlador/ComandoControladorClient.java | CVE-2018-0269<br>CVE-2017-14460 |

Luego del procedimiento explicado previamente, se evalúa el reporte de vulnerabilidades verificadas con respecto a la política establecida y se toma la decisión de romper el build o de continuar con el proceso de CD en el flujo de SecDevOps.

Si la decisión es continuar con el proceso de CD, se exponen los resultados obtenidos. En el resultado del análisis de seguridad en contenedores realizado por Anchore se recopilaron los siguientes hallazgos: para la evaluación de la política se encontraron 181 acciones clasificadas en: 77 con acción parar y 104 como advertencia tomando como acción final detener el flujo. El detalle se observa en la Figura 3-75 y Figura 3-76.


**Figura 3-75.** Resumen evaluación política Anchore

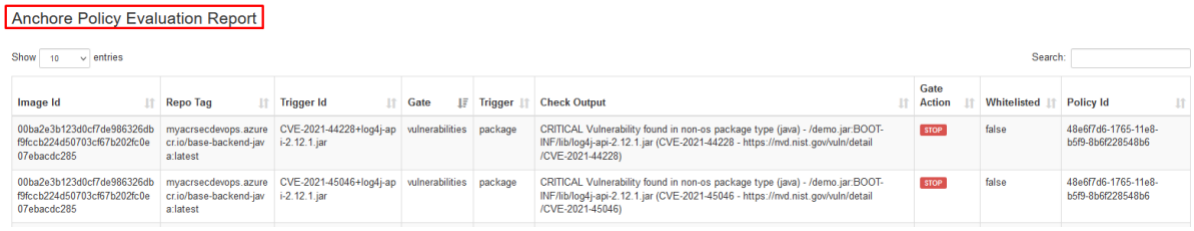



**Figura 3-76.** Reporte evaluación política Anchore

Policy Security

Common Vulnerabilities and Exposures (CVE) List

Show 10 entries Search:

| Tag | CVE ID | Severity | Vulnerability Package | Fix Available | URL |
|---|---|---|---|---|---|
| myacrsecdevops.azurecr.io/base-backend-java:latest | CVE-2019-14697 | Critical | musl-1.1.20-r4 | 1.1.20-r5 | http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-14697 |
| myacrsecdevops.azurecr.io/base-backend-java:latest | CVE-2022-28391 | Critical | busybox-1.29.3-r10 | None | https://nvd.nist.gov/vuln/detail/CVE-2022-28391 |
| myacrsecdevops.azurecr.io/base-backend-java:latest | CVE-2019-19646 | Critical | sqlite-libs-3.26.0-r3 | None | https://nvd.nist.gov/vuln/detail/CVE-2019-19646 |
| myacrsecdevops.azurecr.io/base-backend-java:latest | CVE-2022-1292 | Critical | libcrypto1.1-1.1.1b-r1 | None | https://nvd.nist.gov/vuln/detail/CVE-2022-1292 |
| myacrsecdevops.azurecr.io/base-backend-java:latest | CVE-2021-3711 | Critical | libssl1.1-1.1.1b-r1 | None | https://nvd.nist.gov/vuln/detail/CVE-2021-3711 |
| myacrsecdevops.azurecr.io/base-backend-java:latest | CVE-2021-45046 | Critical | log4j-api-2.12.1 | None | https://nvd.nist.gov/vuln/detail/CVE-2021-45046 |
| myacrsecdevops.azurecr.io/base-backend-java:latest | CVE-2021-44228 | Critical | log4j-to-slf4j-2.12.1 | None | https://nvd.nist.gov/vuln/detail/CVE-2021-44228 |
| myacrsecdevops.azurecr.io/base-backend-java:latest | CVE-2022-27404 | Critical | freetype-2.9.1-r2 | None | https://nvd.nist.gov/vuln/detail/CVE-2022-27404 |
| myacrsecdevops.azurecr.io/base-backend-java:latest | CVE-2019-14697 | Critical | musl-utils-1.1.20-r4 | 1.1.20-r5 | http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-14697 |
| myacrsecdevops.azurecr.io/base-backend-java:latest | CVE-2019-12900 | Critical | libbz2-1.0.6-r6 | 1.0.6-r7 | http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-12900 |

Showing 1 to 10 of 291 entries Previous 1 2 3 4 5 ... 30 Next

**Figura 3-77.** Reporte CVE dado por Anchore

En la evaluación de seguridad con Anchore se detectaron 291 hallazgos clasificados en: 17 críticos, 60 altos, 93 medios y 121 bajos, como se observa en la Figura 3-77. Así mismo, los hallazgos obtenidos por la herramienta kubescape están clasificados en: 4 hallazgos críticos, 9 altos, 17 medios, y 9 bajos, el detalle de lo expuesto se observa en la Figura 3-78, Figura 3-79 y Figura 3-80.

Controls: 57 (Failed: 30, Excluded: 0, Skipped: 9)

| SEVERITY | CONTROL NAME | FAILED RESOURCES | EXCLUDED RESOURCES | ALL RESOURCES | % RISK-SCORE |
|---|---|---|---|---|---|
| Critical | Data Destruction | 21 | 0 | 68 | 31% |
| Critical | Disable anonymous access to Kubelet service | 0 | 0 | 0 | skipped**** |
| Critical | Enforce Kubelet client TLS authentication | 0 | 0 | 0 | skipped**** |
| Critical | Malicious admission controller (mutating) | 2 | 0 | 2 | 100% |
| High | Cluster-admin binding | 3 | 0 | 68 | 4% |
| High | List Kubernetes secrets | 15 | 0 | 68 | 22% |
| High | Malicious admission controller (validating) | 1 | 0 | 1 | 100% |
| High | RBAC enabled | 0 | 0 | 0 | skipped* |
| High | Resources CPU limit and request | 1 | 0 | 2 | 50% |
| High | Resources memory limit and request | 2 | 0 | 2 | 100% |
| High | Workloads with Critical vulnerabilities exposed to external traffic | 0 | 0 | 0 | skipped** |
| High | Workloads with RCE vulnerabilities exposed to external traffic | 0 | 0 | 0 | skipped** |
| High | Workloads with excessive amount of vulnerabilities | 0 | 0 | 0 | skipped** |

**Figura 3-78.** Hallazgos críticos y altos

| | | | | | |
|---|---|---|---|---|---|
| Medium | Allow privilege escalation | 2 | 0 | 6 | 33% |
| Medium | Automatic mapping of service account | 4 | 0 | 4 | 100% |
| Medium | CVE-2022-0492-cgroups-container-escape | 2 | 0 | 2 | 100% |
| Medium | Cluster internal networking | 2 | 0 | 2 | 100% |
| Medium | Configured liveness probe | 2 | 0 | 2 | 100% |
| Medium | CoreDNS poisoning | 7 | 0 | 68 | 10% |
| Medium | Delete Kubernetes events | 6 | 0 | 68 | 9% |
| Medium | Exec into container | 4 | 0 | 68 | 6% |
| Medium | Forbidden Container Registries | 1 | 0 | 2 | 50% |
| Medium | Images from allowed registry | 1 | 0 | 2 | 50% |
| Medium | Ingress and Egress blocked | 2 | 0 | 2 | 100% |
| Medium | Linux hardening | 2 | 0 | 2 | 100% |
| Medium | Namespace without service accounts | 2 | 0 | 4 | 50% |
| Medium | Network mapping | 2 | 0 | 2 | 100% |
| Medium | No impersonation | 4 | 0 | 68 | 6% |
| Medium | Non-root containers | 2 | 0 | 2 | 100% |
| Medium | Portforwarding privileges | 4 | 0 | 68 | 6% |

**Figura 3-79.** Hallazgos medios

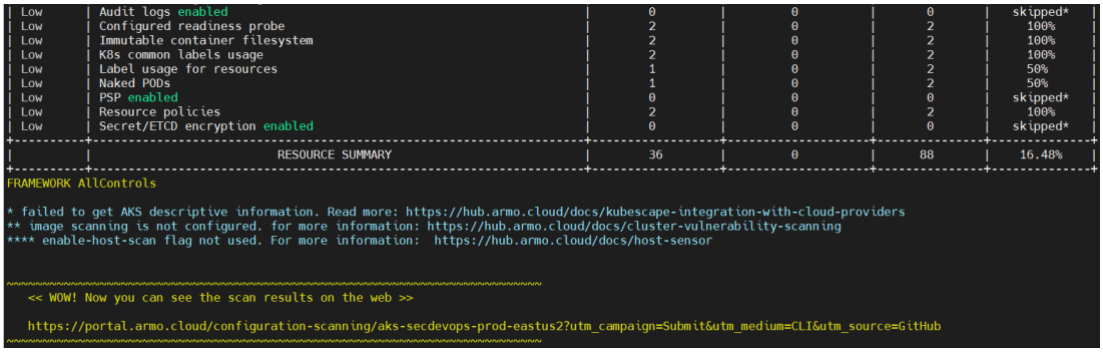



**Figura 3-80.** Hallazgos bajos

Donde en la Figura 3-80 se puede visualizar la URL que direcciona a la consola de kubescape para visualizar y tener una trazabilidad con mayor detalle del análisis, este detalle se observa a continuación en la Figura 3-81.

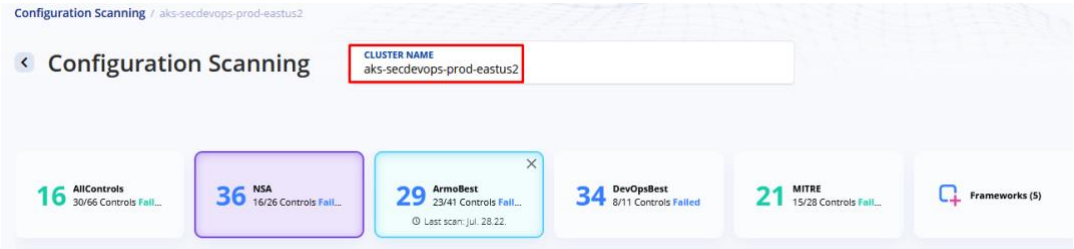



**Figura 3-81.** Reporte CVE dado por Anchore

En este punto la aplicación web fue desplegada en un ambiente pre productivo y se procedió a ejecutar el análisis dinámico de código sobre el endpoint, dicho resultado se muestra a continuación en la Figura 3-82 y Figura 3-83.

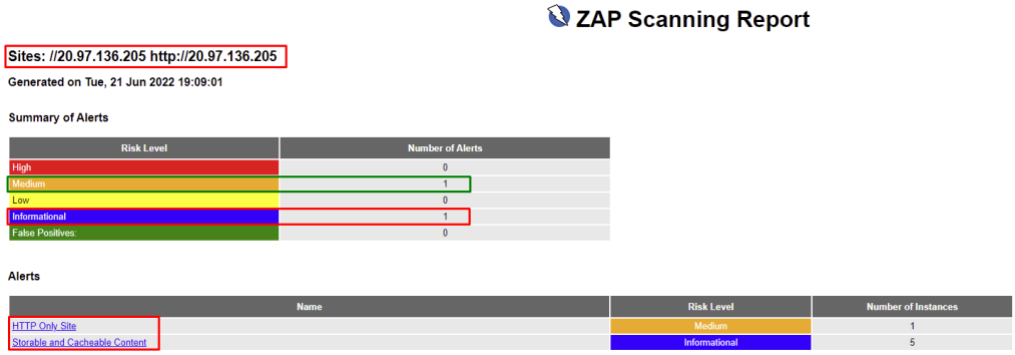



**Figura 3-82.** Reporte DAST

| Medium | HTTP Only Site |
|---|---|
| Description | The site is only served under HTTP and not HTTPS. |
| URL | http://20.97.136.205 |
| Method | GET |
| Parameter | |
| Attack | |
| Evidence | |
| Instances | 1 |
| Solution | Configure your web or application server to use SSL (https). |
| Reference | https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html<br>https://letsencrypt.org/ |
| CWE Id | 311 |
| WASC Id | 4 |
| Plugin Id | 10106 |

**Figura 3-83.** Reporte DAST – Detalle

En este punto, luego de obtener los hallazgos del análisis de seguridad en imágenes Docker, análisis de seguridad en kubernetes y análisis de código dinámico el analista de seguridad dispone del reporte consolidado de todos los hallazgos obtenidos previamente y procede a confirmar si corresponden a una vulnerabilidad o un falso positivo. Para facilidad del ejercicio, en el consolidado de los hallazgos solo se visualizan los calificados con severidad crítica y alta, como se puede visualizar en la Tabla 3-35, Tabla 3-36 y Tabla 3-37

**Tabla 3-35.** Hallazgos en el CD de: Análisis seguridad de imágenes Docker –Prueba 3

| *Nro.* | *Severidad* | *Paquete de vulnerabilidad* | *CVE asociada* |
|---|---|---|---|
| 1 | Críticas | musl-1.1.20-r4 | CVE-2019-14697 |
| 2 | | busybox-1.29.3-r10 | CVE-2022-28391 |
| 3 | | sqlite-libs-3.26.0-r3 | CVE-2019-19646 |
| 4 | | libcrypto1.1-1.1.1b-r1 | CVE-2022-1292 |
| 5 | | libssl1.1-1.1.1b-r1 | CVE-2021-3711 |
| 6 | | log4j-api-2.12.1 | CVE-2021-45046 |
| 7 | | log4j-to-slf4j-2.12.1 | CVE-2021-44228 |
| 8 | | freetype-2.9.1-r2 | CVE-2022-27404 |
| 9 | | musl-utils-1.1.20-r4 | CVE-2019-14697 |
| 10 | | libbz2-1.0.6-r6 | CVE-2019-12900 |
| 11 | | ssl_client-1.29.3-r10 | CVE-2022-28391 |
| 12 | | log4j-api-2.12.1 | CVE-2021-44228 |
| 13 | | sqlite-libs-3.26.0-r3 | CVE-2019-8457 |
| 14 | | libcrypto1.1-1.1.1b-r1 | CVE-2021-3711 |
| 15 | | sqlite-libs-3.26.0-r3 | CVE-2020-11656 |
| 16 | | libssl1.1-1.1.1b-r1 | CVE-2022-1292 |
| 17 | | log4j-to-slf4j-2.12.1 | CVE-2021-45046 |
| 18 | Altas | zlib-1.2.11-r1 | CVE-2018-25032 |
| 19 | | openjdk8-jre-8.212.04-r0 | CVE-2020-14593 |
| 20 | | ssl_client-1.29.3-r10 | CVE-2021-42384 |
| 21 | | snakeyaml-1.25 | GHSA-rvwf-54qp-4r6v |
| 22 | | sqlite-libs-3.26.0-r3 | CVE-2019-19244 |

**Tabla 3-36.** Hallazgos en el CD de: Análisis seguridad en kubernetes –Prueba 3

| *Nro.* | *Severidad* | *Nombre del control* | *Recursos fallados* | *Calificación de riesgo* |
|---|---|---|---|---|
| 1 | Críticas | Data Destruction | 21 | 31% |
| 2 | | Disable anonymous access to Kubelet service | 0 | Omitido |
| | | Enforce Kubelet client TLS authentication | 0 | Omitido |
| 3 | | Malicious admission controller (mutating) | 2 | 100% |
| 4 | Altas | Cluster-admin binding | 3 | 4% |
| 5 | | List Kubernetes secrets | 15 | 22% |
| 6 | | Malicious admission controller (validating) | 1 | 100% |
| 7 | | RBAC enabled | 0 | Omitido |
| 8 | | Resources CPU limit and request | 1 | 50% |
| 9 | | Resources memory limit and request | 2 | 100% |
| 10 | | Workloads with Critical vulnerabilities exposed to external traffic | 0 | Omitido |
| 11 | | Workloads with RCE vulnerabilities exposed to external traffic | 0 | Omitido |
| 12 | | Workloads with excessive amount of vulnerabilities | 0 | Omitido |

**Tabla 3-37.** Hallazgos en el CD de: Análisis dinámico de código DAST –Prueba 3

| *Nro.* | *Severidad* | *Nombre* | *Descripción* | *CWE Asociada* |
|---|---|---|---|---|
| 1 | Media | HTTP Only Site | El sitio está publicado bajo HTTP y no HTTPS | CWE-311 |

De acuerdo con los hallazgos obtenidos en cada uno de los pasos de seguridad llevados a cabo en la ejecución del pipeline y de los puntos donde se evalúa el parámetro RB para continuar o detener el flujo. Se observa la inclusión de controles de seguridad automatizados a lo largo del proceso CI y CD del flujo de DevOps y por el contrario no es necesario esperar al despliegue del aplicativo en producción para realizar los diferentes análisis de seguridad. Dicho lo anterior, se puede mencionar que la estrategia de seguridad propuesta además de lo expuesto previamente ofrece la integración de controles de seguridad adicionales y la ventaja de ser adaptable a la madurez del equipo que la desee implementar.

# 4. Conclusiones y Recomendaciones

## 4.1. Conclusiones

Basados en las diferentes fuentes bibliográficas y comunidades de desarrollo reconocidas a nivel de mundial, se concluye que el lenguaje de programación web más popular en el momento del desarrollo de esta investigación es Java.

A partir de la revisión del estado del arte, se evidenció carencia de trabajos relacionados con la metodología, cultura e implementación de seguridad en DevOps. La escasez de estrategias y/o recomendaciones de seguridad fue notable, en consecuencia, la estrategia de seguridad expuesta en el desarrollo de esta tesis adoptó gran parte de lo ofrecido por la literatura para desarrollar puntos importantes adicionales, tales como: adopción del parámetro RB, sugerencia de nuevas estrategias de seguridad tanto en CI como en CD a las comunes y, adicional a esto, parámetros transversales para lograr la adopción de la metodología integral de SecDevOps.

Se construyó una estrategia de seguridad integral, flexible y adaptable, tanto al tamaño de la empresa, como al nivel de madurez del equipo de seguridad que la desee implementar. La estrategia está basada en siete parámetros clasificados en obligatorios y transversales, los cuales cubren la totalidad de puntos necesarios para cumplir con los criterios de una estrategia integral. Apoyados en esto y con el prerrequisito de emplear herramientas de software libre y servicios ofrecidos por Azure, se construyó la estrategia. A pesar de esto, se evidenció que fue necesaria la adopción de conocimientos técnicos transversales para la interacción y despliegue de las plataformas. Por ello, de acuerdo con la madurez, capacidad del equipo y presupuesto, la estrategia desarrollada cuenta con la posibilidad de ser adaptada fácilmente a herramientas pagas para su instalación e implementación.

A pesar de la carencia de información obtenida de la revisión de la literatura, la estrategia desarrollada a lo largo del documento ofrece esa guía técnica para apoyar a una persona y/o equipo que cuente con conocimiento técnico estándar en DevOps para lograr la adaptación de acuerdo con su madurez.

Según los resultados obtenidos en la ejecución de cada una de las pruebas, se concluye que la metodología DevOps no considera la implementación de seguridad a lo largo del flujo, por eso, es

muy probable que en la puesta en producción el aplicativo contenga brechas de seguridad, lo que conlleva a la ejecución de reprocesos para la corrección de vulnerabilidades encontradas. En la prueba dos se encontró que la seguridad estándar al final del proceso CI y proceso CD en el flujo DevOps, es la forma más común que emplean las empresas de desarrollo para implementar seguridad en la construcción y despliegue del aplicativo, lo cual también incurre en reprocesos, debido a que las pruebas realizadas son ejecutadas por el departamento aparte del de desarrollo de la compañía o tercerizadas, lo cual ralentiza el proceso de la aplicación de la metodología DevOps. Por último, en la prueba tres se confirmó que la estrategia de seguridad propuesta incrementa los puntos para evaluar la seguridad mediante la automatización de dichas estrategias de seguridad y así evitar el despliegue del aplicativo, mitigando en buena medida los riesgos de seguridad.

Para romper el build en la metodología SecDevOps propuesta, es necesario contar con la intervención del analista de seguridad para la verificación de los hallazgos, dado que, hasta el desarrollo de este trabajo investigativo, no se cuenta con una herramienta para automatizar la verificación de los hallazgos como vulnerabilidades.

## 4.2. Recomendaciones

Teniendo en cuenta que el resultado de cada herramienta de seguridad suministra una cantidad considerable de hallazgos, es necesario realizar una validación de cada uno de estos para verificar si es una vulnerabilidad o falso positivo, por lo que es requerido probar cada hallazgo, demandando un gran esfuerzo y generando tiempos altos en la entrega del reporte final. Basados en esto, se sugiere trabajar en una estrategia de automatización que facilite y agilice la verificación para lograr una respuesta más favorable para el equipo de SecDevOps.

Como alcance de esta tesis, se definió el lenguaje de programación Java para aplicaciones web, pero es claro que en el mercado existen otros lenguajes de programación que son adoptados y empleados por empresas de desarrollo, tales como: php, C#, Python, entre otros. Por ende, es recomendable, apoyados en los insumos construidos y ofrecidos a lo largo de este documento, realizar una adaptación al lenguaje requerido mediante la modificación de algunos recursos empleados para el lenguaje de programación Java en esta investigación.

Dentro de los parámetros transversales, apoyados en la estrategia SecDevOps, se mencionó la importancia del monitoreo continuo de seguridad. Como trabajo futuro se recomienda construir una estrategia de monitoreo continuo y la selección de una herramienta para su implementación, que permita disponer de la trazabilidad y visualización del estado de las vulnerabilidades respecto de la política definida a lo largo de todo el flujo DevOps.

# A. Anexo: Revisión estado de la literatura

En este anexo se expone el resultado de la revisión del estado de la literatura. De acuerdo al procedimiento de investigación aplicado. La búsqueda bajo los criterios de inclusión y definidos arroja 95 artículos, luego de realizar la valoración de calidad quedan 22 artículos de los cuales 13 artículos son tomados en el estado del arte. A continuación, en la Tabla 4-1 y Tabla 4-2 se expone con más detalle los artículos seleccionados.

**Tabla 4-1.** Artículos iniciales, artículos luego de la valoración de la calidad

| Paso de la revisión de la literatura | Cantidad de artículos |
|---|---|
| Búsqueda definiendo los criterios de exclusión e inclusión | 95 |
| Valoración de la calidad | 22 |

En la Tabla **4-2** se ofrece un detalle de los 22 artículos que pasaron el filtro de valoración de calidad, los artículos subrayados en verde claro son los artículos empleados para desarrollar el estado del arte de la tesis.

**Tabla 4-2.** Detalle artículos valoración de la calidad

| Nro. | Nombre artículo | Fuente | Año | Autor | Incluidos en las referencias |
|---|---|---|---|---|---|
| **1** | DevSecOps: integración de herramientas sast, dast y de análisis de docker en un sistema de integración continua. | Universidad Oberta de Catalunya | 2019 | Caño Quintero José Joaquín | SÍ |
| **2** | Securing a deployment pipeline | IEEE | 2015 | Len Bass, Ralph Holz, Paul Rimba, An Binh Tran, Liming Zhu | SÍ |
| **3** | Seguridad en el ciclo de vida del desarrollo del software. DevSecOps | Universidad Oberta de Catalunya | 2019 | Ricote Iris Sergio | SÍ |
| **4** | Vulnerabilities in continuous delivery pipelines? a case study | IEEE | 2019 | Paule Christina, F. Dullmann Thomas, van Hoorn André | SÍ |
| **5** | Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices | IEEE ACCESS | 2017 | MOJTABA SHAHIN, MUHAMMAD ALI BABAR, AND LIMING ZHU | NO |
| **6** | A prologue of jenkins with comparative scrutiny of various software integration tools | Amity University | 2015 | Rai Preeti, Madhurima, Dhir Saru, Madhulika, Garg Anchal | NO |

| | | | | | |
|---|---|---|---|---|---|
| **7** | From continuous integration to continuous assurance | University of Wisconsin | 2018 | A. Kupsch James, P. Miller Barton, Basupalli Vamshi, Burger Josef | NO |
| **8** | Software process improvement: continuous integration and testing for web application development | University of Tampere | 2009 | Muhonen Matias | NO |
| **9** | Docker ecosystem - vulnerability analysis | Computer Communications | 2018 | Martin A, Rasponi S, Combe T, Di Petro R. | NO |
| **10** | Securing devops - detection of vulnerabilities in cde pipelines | University of Stuttgart | 2018 | Paule Christina | SÍ |
| **11** | Security for devops deployment processes: defenses, risks, research directions | Department of Computer Science, University of West Florida, Pensacola, Florida, USA | 2016 | Wilde Norman, Eddy Brian, Patel Khyati, Cooper Nathan, Gamboa Valeria, Mishra Bhavyansh, Shah Keenal | SÍ |
| **12** | BP: security concerns and best practices for automation of software deployment processes an industrial case study | IEEE Secure Development Conference | 2018 | Mohan Vaishnavi, Othmane Lotfi Ben, Kres Andre | NO |
| **13** | Software security in devops: synthesizing practitioners' perceptions and practices | International Workshop on Continuous Software Evolution and Delivery - North Carolina State University | 2016 | Ur Rahman Akond Ashfaque. Wiiliams Laurie | NO |
| **14** | Seguridad en docker | Universidad Oberta de Catalunya | 2018 | Fernández Ginés Xavier | SÍ |
| **15** | A framework for detecting and preventing security vulnerabilities in continuous integration/continuous delivery pipelines | University of Twente | 2019 | Koopman Michael | SÍ |
| **16** | Análisis estático de vulnerabilidades en kubernetes para entornos de integración continua | Universidad Oberta de Catalunya | 2020 | Aladrén García Julio | SÍ |
| **17** | A devops framework for quality-driven self-protection in web software systems | Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering | 2018 | N. Beigi-Mohammadi, M. Litoiu, M. Emami-Taba, L. Tahvildari, M. Fokaefs, E. Merlo y I. V. Onut | SÍ |
| **18** | Security support in continuous deployment pipeline | International Conference on Evaluation of Novel Approaches to Software Engineering, ENASE. | 2017 | F. Ullah, M. Shahin, M. Zahedi y M. A. Babar | SÍ |
| **19** | Docker container security in cloud computing | IEEE | 2020 | S. M. T. N. J. C. Kelly Brady | SÍ |
| **20** | Seguridad en docker - hids | Universidad Oberta de Catalunya | 2018 | Fernández Gamboa Juan | NO |
| **21** | Herramientas de prueba de seguridad de aplicaciones | Universidad Oberta de Catalunya | 2017 | Gómez Zafra Gina Alexandra | SÍ |
| **22** | DevSecOps: enabling security by design in rapid software development | Universiteit Utrecht | 2019 | Amr Mustafa Ahmed | NO |

# B. Anexo: Metodologías de desarrollo de software clásicas, ágiles, y seguras

En este anexo se desarrollan y se exponen las diferentes metodologías de desarrollo de software, clásicas, ágiles y seguras que se han propuesto para la construcción de aplicativos.

**Metodologías para el desarrollo de software clásicas**

Las metodologías tradicionales imponen una disciplina de trabajo sobre el proceso de desarrollo del software, para ello, se hace énfasis en la planificación total de todo el trabajo a realizar y una vez que está todo detallado, comienza el ciclo de desarrollo del producto software. Se centran especialmente en el control del proceso, mediante una detallada definición de roles, actividades, herramientas, modelado y documentación. Además, las metodologías tradicionales no contemplan los cambios durante el proceso, por lo que no son las más recomendadas cuando se trabajan en un entorno donde los requerimientos no pueden predecirse o puedan variar en el tiempo [128].

- ***Ciclo de vida del desarrollo de software***

SDLC, System Development Life Cycle. Las tareas establecidas en este marco se definen en cada paso del proceso de desarrollo de software. Consiste en un plan detallado que describe como desarrollar, mantener y reemplazar software específico, mejorando la calidad del software y el proceso general del desarrollo, bajo las fases de: planificación, implementación, pruebas, documentación, despliegue y mantenimiento [129].

- ***Incremental***

Permite construir el proyecto en etapas incrementales en donde cada etapa agrega funcionalidad y valor. Las etapas contempladas en esta metodología son: requerimientos, diseño, codificación, pruebas y entrega. En comparación al modelo de cascada, este permite entregar al cliente un producto más rápido, reduciendo los riesgos gracias a la visibilidad sobre el progreso de las nuevas versiones. Además, proporciona retroalimentación que permite atacar los riesgos más críticos desde el principio, ejecución de pruebas, e integraciones constantes [6].

- ***Prototipos***

Un prototipo es una versión preliminar de un sistema de información con fines de demostración o evaluación. El prototipo de requerimientos es la relación de una implementación parcial de un sistema, para el propósito explícito de aprender sobre los requerimientos del sistema. El prototipo puede ser usado como parte de la fase de requerimientos o justo antes de la fase de seguimiento. Muchos usuarios y clientes encuentran que es mucho más fácil proveer retroalimentación convenientemente basada en la manipulación, diferente del modelo evolutivo, donde los requerimientos mejor entendidos están incorporados; un prototipo se construye, generalmente, con los requerimientos entendidos más pobremente. Se compone por: recolección y refinamiento de requisitos, diseño rápido, construcción del prototipo, evaluación del prototipo por el cliente, refinamiento del prototipo, producto de ingeniería [6].

- ***Cáscada***

Waterfall. Es el enfoque metodológico que ordena rigurosamente las etapas del proceso para el desarrollo de software, de tal forma que el inicio de cada etapa debe esperar la finalización de la etapa anterior. Al final de cada etapa, se realiza una revisión en la cual se determina si el proyecto en dicha fase sí cumplió los objetivos y puede avanzar a la siguiente fase. Este modelo fue el primero en originarse y es la base de los modelos de ciclo de vida posteriores a este. Uno de los inconvenientes de esta metodología es que no se puede esperar a que las especificaciones iniciales sean correctas, completas y el usuario cambie de opinión o requiera reconfiguración sobre alguna característica. Los pasos que componen esta metodología son: ingeniería y análisis del sistema, análisis de los requisitos, diseño, codificación, prueba, y mantenimiento [6].

- ***Espiral***

Esta metodología suma las ventajas de las metodologías de desarrollo en cascada y prototipos, apoyándose en el concepto de análisis de riesgo. Está compuesta por cuatro actividades: planificación, análisis de riesgo, ingeniería y evaluación del cliente. En cada fase el producto gana “madurez” (aproximación al final deseado), hasta que en una iteración se logre el objetivo deseado y este sea aprobado, se terminan las iteraciones [6].

**Metodologías para el desarrollo de software ágiles**

El enfoque de desarrollo clásico se enfatiza en el control del proceso mediante definición de roles, actividades, artefactos, en los cuales se incluye un modelado y documentación detallada. El esquema tradicional ha demostrado ser efectivo y aplicado en proyectos grandes, sin embargo, este enfoque clásico no es el más adecuado ni óptimo para proyectos donde el sistema es muy cambiante y es necesario reducir tiempos en el desarrollo sin disminuir la calidad sobre este. Dada las desventajas en cuanto a tiempo y flexibilidad, se deja al lado el buen hacer de la ingeniería del software asumiendo el riesgo que esto conlleva. La metodología necesariamente ha de ser ágil, debe contar con ciclos cortos de desarrollo y, si es necesario, se pueden incrementar las funcionalidades en cada iteración del mismo. Bajo este entorno y esta necesidad han nacido las metodologías ágiles y como consecuencia, se creó el manifiesto ágil [6].

Es así entonces que el enfoque ágil para el desarrollo de software busca distribuir de forma permanente sistemas de software en funcionamiento diseñados con iteraciones rápidas. En concreto, el desarrollo ágil busca proporcionar en poco tiempo piezas pequeñas de sistemas de software en funcionamiento para mejorar la satisfacción del cliente. Por lo general, el desarrollo ágil de software implica que haya grupos pequeños de desarrolladores de software y otra figura que se encarga de reunir los integrantes del grupo durante el ciclo de vida del desarrollo de software. La metodología ágil favorece un enfoque sencillo de la documentación de software, y acepta los cambios que puedan surgir en las diferentes etapas del ciclo de vida, en lugar de resistirse a ellos [130].

- ***Programación Extrema (XP, Extreme Programming)***

XP, Extreme Programming. La programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Los cambios de requisitos que ocurran durante el proceso durante el proceso, son un aspecto normal de la metodología; estos pueden ser tomados o no dependiendo del valor agregado que le aporten al proyecto en esa etapa. Se centra en mejorar y potenciar las relaciones interpersonales, promoviendo trabajo en equipo, motivando al aprendizaje y trabajando para favorecer un buen clima laboral. Se tienen tres (3) categorías, en las cuales se agrupan las doce (12) prácticas de XP:

- ✓ Metodología de programación: diseño sencillo, pruebas (testing), refactorización, codificación con estándares.
- ✓ Metodología de equipo: propiedad colectiva del código, programación en parejas, integración continua, entregas semanales, e integridad con el cliente.
- ✓ Metodología de procesos: clientes en sitio, entregas frecuentes y planificación.

Fases del ciclo de vida de la programación extrema:

Exploración – Planificación – Iteración 1…N – Producción – Mantenimiento – Muerte [6].

- ***Scrum***

Marco de trabajo por el cual las personas pueden abordar problemas complejos adaptativos, a la vez que entregar productos del máximo valor posible productiva y creativamente. Scrum muestra la eficacia de las técnicas de gestión de producto y las técnicas de trabajo de modo que podamos mejorar continuamente el producto, el equipo y el entorno de trabajo. El marco de trabajo scrum consiste en los equipos scrum, sus roles, eventos, artefactos y reglas asociadas, donde cada componente cumple con una tarea específica y todos son esenciales para el éxito de scrum. La esencia de scrum está basada en la construcción de equipos pequeños, altamente flexibles y adaptativos. A continuación, se mencionan algunos componentes importantes dentro de scrum:

*Equipo Scrum:* dueño de producto (Product Owner), equipo de desarrollo (Development Team), y scrum master.

*Eventos de Scrum:* sprint, planning, daily, review, retrospective.

*Artefactos de Scrum*: lista de producto (Product Backlog), lista de pendientes del sprint (Sprint Backlog) [131]. A continuación, en la Figura 4-1 se muestra el flujo de la metodología scrum;

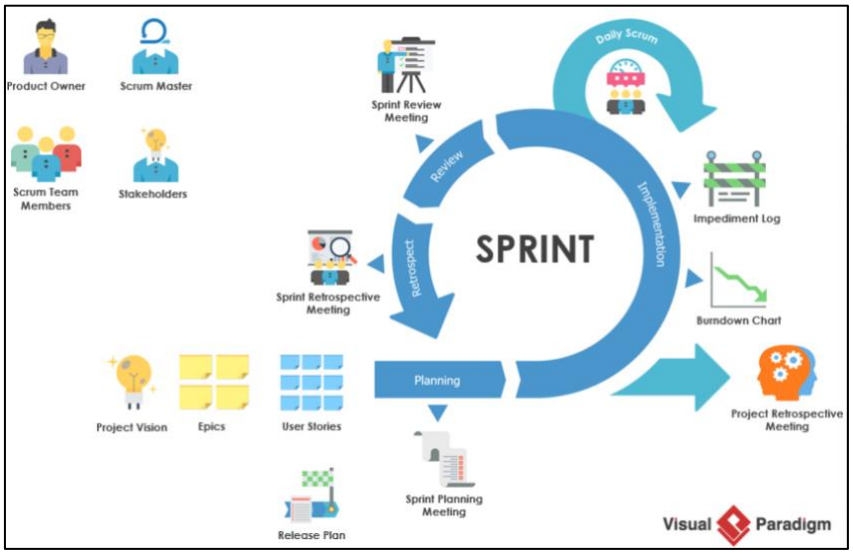



**Figura 4-1.** Flujo de la metodología SCRUM **[132]**

- ***Crystal***

Se trata de un conjunto de metodologías para el desarrollo de software caracterizadas por estar centradas en las personas que componen el equipo y la reducción al máximo del número de artefactos producidos. El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipo o definidas. *Crystal Clear* es la encarnación más ágil de la serie y de la que más documentación se dispone. La misma se define con mucho énfasis en la comunicación y de forma muy liviana en relación a los entregables. Crystal maneja iteraciones cortas con feedback (retroalimentación) frecuente por parte de los usuarios/clientes, minimizando de esta forma la necesidad de productos intermedios. Como características importantes se tienen: la entrega frecuente, comunicación, información relevante, seguridad personal, focalización, facilidad en la comunicación con usuarios expertos [6].

- ***Kanban***

Se basa en la idea de que el trabajo en curso debería limitarse y solo se debe empezar con la nueva funcionalidad cuando la funcionalidad anterior haya sido entregada. Se hace uso de un panel electrónico de tarjetas (tablero kanban) para gestionar el flujo de trabajo y asignaciones; con este punto se da un aporte proporcionando transparencia al proceso, evitando cuellos de botella, desperdicios a lo largo del tiempo y todo lo que pueda impactar en el rendimiento de la

organización. Proporciona a los miembros una mayor visibilidad sobre los efectos de sus acciones o falta de acción, como resultado, se evidencia la evolución incremental en los procesos existentes. Una de las principales ventajas es la facilidad de aplicar, actualizar y adaptar por parte del equipo. Además, la técnica altamente visual de gestión de las tareas, permite ver el estado de los proyectos, y la posibilidad de pautar el desarrollo del trabajo de manera efectiva. Para la aplicación de la metodología kanban es necesario conocer los siguientes aspectos: definir el flujo de trabajo de los proyectos, visualizar las fases del ciclo de producción, "Stop Starting" y el control de flujo [6].

- ***Desarrollo rápido de aplicaciones***

RAD, Rapid Application Development. El modelo de desarrollo rápido de aplicaciones de software se enfoca en el ciclo de diseño de usuario en la creación de prototipos, pruebas y refinamiento. RAD, con respecto al método de cascada (waterfall), es flexible y responde a las entradas del usuario. A pesar de que el modelo cuente con 30 años desde su primera implementación, se ha tenido en cuenta en empresas modernas, ya que proporciona a los usuarios experiencias atractivas en todos los dispositivos, desde aplicaciones nativas móviles, web, hasta aplicaciones de conversación. Las cuatro (4) fases del ciclo de vida de RAD son: planear, diseño informado por el usuario, construcción, paso a producción [133].

- ***Desarrollo basado en funcionalidades***

FDD, Feature Driven Development. Como las otras metodologías adaptables, se enfoca en iteraciones cortas que entregan funcionalidad tangible. Sin embargo, fue diseñado para trabajar con otras actividades de desarrollo de software y no requiere la utilización de ningún modelo de proceso específico. Hace énfasis en aspectos de calidad durante todo el proceso e incluye un monitoreo permanente del avance del proyecto. Una característica o (feature) se basa en un ciclo muy corto de iteración, nunca supera las dos semanas y el análisis y desarrollo están orientados en cumplir un listado de features. Esta metodología sigue cinco (5) fases iterativas: desarrollo o modificación de un modelo global, creación o modificación de la lista de features, planificación, diseño, implementación [6].

- ***Desarrollo adaptivo de software***

ASD, Adaptive Software Development. La idea principal de esta metodología parte de que las necesidades del cliente siempre son cambiantes durante y posteriormente a la entrega del

desarrollo del proyecto. El punto diferenciador de esta metodología es que crea una cultura adaptiva mediante métodos y técnicas para adaptarse fácilmente al cambio y no luchar contra él. Ofrece un ciclo de vida iterativo y cada ciclo puede ser modificado al paralelo de la ejecución de otro ciclo, este ciclo usado por ASD es conocido como: especular-colaborar-aprender [6].

***Desarrollo fino, desarrollo de software fino***

LD, Development. LSD Lean Software. La filosofía "lean" consiste en tener un equipo muy preparado, altamente motivado y muy unido. Los activos más importantes a tener en cuenta cuando se está desarrollando un proyecto bajo Lean Development no son el tiempo o el dinero que se están invirtiendo, sino el grado de compromiso y, sobre todo, cuánto está aprendiendo el equipo. Para proyectos a medio plazo, este método es muy eficaz, ya que concibe una idea, se programa, se lanza un prototipo que se prueba y se analiza el comportamiento, luego del resultado del análisis y las pruebas, se toma la decisión de si cumple el objetivo o es necesario cumplir con otro ciclo iterativo para refinar el prototipo. Luego de varias iteraciones se tendrá un prototipo muy definido, el cual cumple con el objetivo con el que fue concebido. Dentro de los principios de lean se tiene: eliminar los desperdicios, ampliar el aprendizaje, decidir lo más tarde posible, reaccionar tan rápido como sea posible, potenciar el equipo, crear la integridad [6].

***Proceso Unificado de Desarrollo de Software***

RUP, Rational Unified Process. Metodología de desarrollo de software que está basada en componentes e interfaces bien definidas y junto con el Lenguaje Unificado de Modelado (UML), constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización. Cuenta con las fases: concepción o inicio, elaboración, transición [6].

**Metodologías para el desarrollo seguro de software**

Además de mencionar las metodologías clásicas y ágiles para el desarrollo de software, a continuación, se hace una breve reseña de las diferentes metodologías seguras que existen y que actualmente son usadas en el mercado para la construcción de software.

- ***Ciclo de vida de desarrollo seguro***

S-SDLC, Secure System Development Life Cycle. Colección de buenas prácticas enfocadas en agregar seguridad a la metodología SDLC. La implementación de seguridad requiere un esfuerzo al deber ser implementada en cada una de las fases del SDLC, desde la recopilación de requisitos, hasta la implementación y mantenimiento; además, el equipo de desarrollo debe cambiar de mentalidad y, aparte de preocuparse por la funcionalidad, también se deberá preocupar por la seguridad. Esto reduce el riesgo de encontrar vulnerabilidades de seguridad en la aplicación cuando sale a producción y así minimizar el impacto cuando se encuentren en las fases finales del ciclo. Las fases del ciclo S-SDLC son:

- Requerimientos: recolección de requerimientos y cumplimientos de seguridad.
- Diseño: uso de la privacidad al diseñar, evaluación de riesgos.
- Desarrollo: pruebas de seguridad en código (SAST - SCA), pruebas unitarias, revisión de código por pares.
- Verificación: pruebas funcionales automatizadas, pruebas de penetración.
- Mantenimiento y evolución: plan de restablecimiento (rollback), creación de un plan de respuesta a incidentes [134].

- ***Ciclo de vida de desarrollo seguro de Microsoft***

SDL, Microsoft Security Development LifeCylce. Proceso que Microsoft ha adoptado para el desarrollo de software que abarca la adición de una serie de actividades centradas en la seguridad y entregables a cada una de las fases del proceso de desarrollo de software de Microsoft. Estas actividades incluyen el desarrollo de modelos de amenazas durante el diseño de software, el uso de herramientas de análisis de código de análisis estático durante la implementación y la realización de revisiones de código y pruebas de seguridad en las últimas fases. Antes de que el software sujeto al SDL pueda ser lanzado, debe ser sometido a una revisión final de seguridad por un equipo independiente de su grupo de desarrollo. En comparación con el software que no ha estado sujeto a la SDL, el software que ha sufrido la SDL ha experimentado una tasa significativamente reducida de descubrimiento externo de vulnerabilidades de seguridad. Para la implementación de las diferentes actividades de seguridad, se cuenta con las siguientes fases: requerimientos, diseño, implementación, verificación, liberación [135].

- ***Corrección por construcción***

CbyC, Correctness by Construction. Método radical, efectivo y económico para construir software con integridad demostrable para aplicaciones críticas y de seguridad. CbyC combina los métodos formales y el desarrollo ágil. Está basado en tres principios simples: que el encontrar errores se dificulte, eliminar los errores lo más cerca al punto de inicio, generar evidencia de idoneidad para el propósito a lo largo del desarrollo. A diferencia de compilar y depurar, CbyC busca que el producto sea correcto desde el inicio de su construcción, adaptándose a los objetivos de construcción de seguridad recomendados. Es compatible con los principios de PSP y TSP, y si se combina con estos, puede haber una tasa de errores menor. Para lograr los objetivos, se implementan las siguientes técnicas que son los bloques de construcción del proceso CbyC: notaciones sonoras, validación fuerte, desarrollo incremental, evitar la repetición, luchar por la simplicidad, gestión del riesgo, pensamiento activo [136].

- ***Proceso integral y ligero de seguridad de aplicaciones***

CLASP, Comprehensive, Lightweight Application Security Process- Conjunto de componentes de proceso basado en roles y en la actividad, guiado por las mejores prácticas formalizadas. CLASP está diseñado para ayudar a los equipos de desarrollo de software a desarrollar la seguridad en las primeras etapas de los ciclos de vida de desarrollo de software existentes y nuevos en una forma estructurada, repetible y medible. Se basa en un extenso trabajo de campo realizado por desarrolladores de software seguro, en los cuales se creó un conjunto integral de requisitos de seguridad. Estos requisitos resultantes forman la base de las mejores prácticas de CLASP, que pueden permitir a las organizaciones abordar sistemáticamente las vulnerabilidades. Componentes del proceso CLASP:

- ✓ Vistas CLASP (5): vistas para dar un entendimiento a gran escala del proceso CLASP, incluye cómo los componentes interactuan y cómo se aplican a cierto ciclo de vida del desarrollo de software.
- ✓ Mejores prácticas CLASP (7): son la base de todas las actividades relacionas para el desarrollo de software, ya sea planificando, diseñando o implementando, incluyen además el uso de todas las herramientas y técnicas que soporta CLASP.

- ✓ 24 Actividades CLASP: diseñadas para permitir una fácil integración entre las actividades de seguridad y el SDL; se relacionan las mejores prácticas con las actividades y los roles de los proyectos.
- ✓ Recursos CLASP (Se incluyen las palabras claves CLASP): ayudan a la planificación, ejecución y cumplimiento de las actividades.
- ✓ Taxonomía CLASP: clasificación de alto nivel de 104 tipos de problemas o vulnerabilidades, en el código fuente divididos en cinco (5) categorías de alto nivel [137].

- ***Equipo de procesos de software***

TSP, Team Software Process. Metodología para administrar el trabajo de mejora y el desarrollo de los procesos de software, garantizando un entorno de trabajo agradable y natural para los equipos. Brinda unos pasos estructurados que indican que se debe hacer en cada fase del desarrollo del proyecto y muestra la conexión de cada fase para construir un producto completo, sin olvidarse de cómo conformar un equipo capaz de lograr los objetivos en el desarrollo de software de calidad. Las fases del ciclo de vida de TSP son: lanzamiento, estrategia, planeación, requerimientos, diseño, implementación, pruebas, postmorten [138].

- ***Garantía de seguridad del software Oracle***

OSSA, Oracle Software Security Assurance. Abarcando cada fase del ciclo de vida del desarrollo del producto, Oracle Software Security Assurance (OSSA) es la metodología de Oracle para crear seguridad en el diseño, el desarrollo, la prueba y el mantenimiento de los productos ya sea que los clientes los usen en las instalaciones o se entreguen a través de Oracle Cloud. El objetivo de Oracle es garantizar que sus productos ayuden a los clientes a cumplir con sus requisitos de seguridad. Oracle Software Security Assurance es un conjunto de normas, tecnologías y prácticas líderes en el sector que tiene el siguiente objetivo:

- ✓ **Promover innovaciones de seguridad:** Oracle ofrece una larga serie de innovaciones de seguridad, a implementar en las diferentes soluciones posibles.
- ✓ **Reducir la incidencia de las deficiencias de seguridad en todos los productos de Oracle:** en los diferentes programas del OSSA se incluyen estándares de codificación seguridad, entrenamiento para el desarrollo seguro, formación de seguridad en los grupos y el uso de herramientas automatizadas.

- ✓ **Reducir el impacto que tienen sobre los clientes las deficiencias de seguridad de los productos lanzados:** tratamiento hacia los clientes para así brindar la mejor experiencia en la actualización de seguridad [139].

- ***Orientación adecuada y eficaz en seguridad de la información***

AEGIS, Appropriate and Effective Guidance in Information Security. Es un proceso de desarrollo de software para sistemas seguros y usables, integrado con el proceso incremental de desarrollo. Los pasos que describen el núcleo de AEGIS consisten en la identificación y el aseguramiento de los participantes los miembros del equipo, para así luego modelar los activos del sistema usando la semántica “UML", donde se debe asignar un valor a esos activos y finalizar con el diseño de contramedidas en torno al análisis de riesgos realizado sobre estos activos. Las necesidades se direccionan gracias a la participación de los usuarios en el diseño de seguridad, junto con la consideración de que se está dando al contexto del usuario durante el modelamiento de los requerimientos de seguridad y el diseño de las contramedidas. En esta metodología, el accionista (stakeholder) está envuelto activamente en todo el proceso de elección de los requerimientos de seguridad y decisión en las contramedidas de seguridad, dado que estos cuentan con un dominio de conocimiento pertinente inicial. Para el desarrollo del proceso se cuenta con las siguientes actividades, las cuales componen todo el flujo de trabajo de esta metodología: reunir participantes, identificar y modelar los activos en el contexto, valorar activos de acuerdo a la seguridad, análisis de riesgos y diseño de seguridad [140].

- ***Proceso racional unificado seguro***

RupSec, Rational Unified Process-Secure. "RUP" es uno de los más completos modelos de procesos flexibles, configurable, fácil para entender y la mayoría de las pautas y actividades están basadas en estándares de ingeniería de software propuestos por la ISO e IEEE. El principal propósito de RupSEC es definir el modelo de proceso del software en el cual todos los requerimientos de seguridad son considerados en todas las fases del desarrollo de software, dichas fases se componen de: modelamiento del negocio, requisitos, análisis, diseño, pruebas, configuración y gestión de cambios, gestión de proyectos y ambientes. RupSec ha sido desarrollado para proponer nuevas extensiones en las fases y disciplinas de RUP en forma de roles, actividades y artefactos [141].

- ***Modelo de desarrollo de software seguro***

SSDM, Secure Software Development Model. Es un modelo unificado que integra la seguridad con la ingeniería de software para así garantizar una producción eficaz y a la vez segura. El modelo combina algunas técnicas existentes de seguridad de software, las cuales están enfocadas en garantizar un software seguro; además, en el modelo se combina el enfoque del camino de ingeniería tradicional con técnicas de ingeniería en seguridad para así garantizar un producto fiable y seguro. El camino de ingeniería de seguridad se divide en cinco (5) etapas: entrenamiento de seguridad, modelamiento de amenazas, especificaciones de seguridad (SS), revisión de las especificaciones de seguridad y, para finalizar, unas pruebas de penetración. Sumado a esto, se mencionan las etapas del desarrollo de software: Definición de requisitos, diseño, codificación, sistema de pruebas, implementación y mantenimiento [142].

# C. Anexo: Instalación de las herramientas para la plataforma DevOps propuesta

Se expone el paso a paso seguido para el despliegue de cada una de las herramientas que componen la plataforma DevOps.

En el desarrollo de este anexo se describe el paso a paso de la configuración e instalación de cada una de las herramientas necesarias para construir la plataforma DevOps. Además, el despliegue de cada una de las herramientas se realiza en Azure.

Partiendo del diseño de la plataforma DevOps ilustrado en la Figura 3-20, capítulo 3.3.1. Se inicia con la configuración de las redes y subredes base para los otros elementos como se observa en la Figura 4-2.

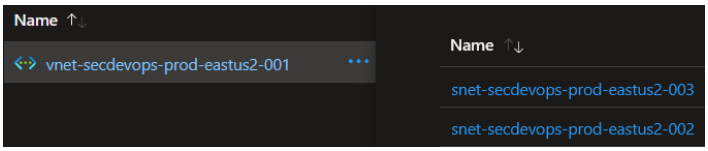



**Figura 4-2.** Creación redes y subredes en Azure

A partir de esto, se despliega la máquina virtual "vm-secdevops-prod-eastus2-001" con sistema operativo Linux, Ubuntu 20.04 LTS siguiendo el paso a paso estándar desde la consola de Azure. Dentro de las características a tener en cuenta son: Subnet asociada, DNS configurado, grupo de seguridad asociado, el tamaño de la máquina virtual, entre otros, como se observa en la Figura 4-3.

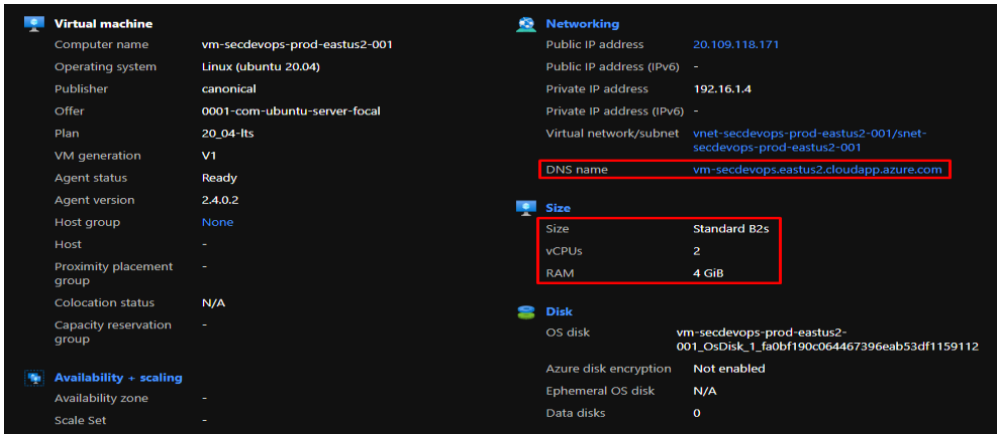



**Figura 4-3**. Características máquina virtual para la ejecución de Jenkins creada en Azure

Para el acceso a la máquina virtual creada se configura una conexión SSH y adicionalmente se adiciona en el grupo de seguridad (nsg) una regla para consumir la máquina virtual desde la IP pública de la estación local, ver Figura 4-4.

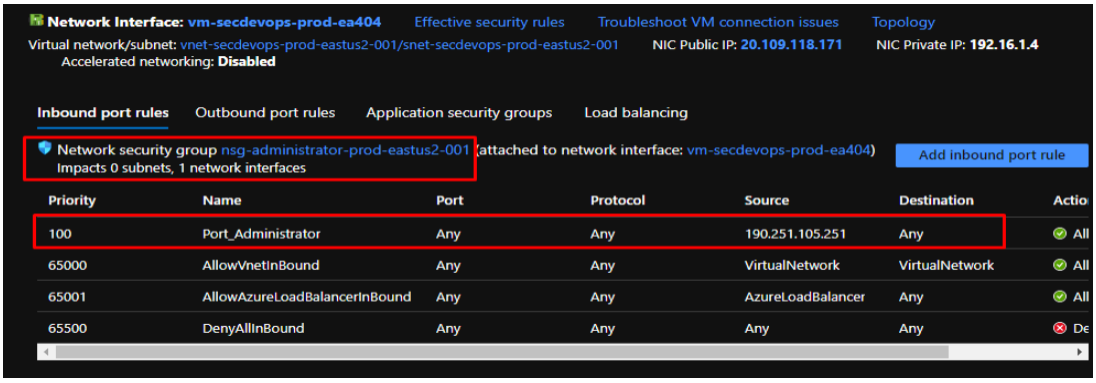



**Figura 4-4.** Creación regla en el grupo de seguridad creado

Mediante la conexión SSH creada previamente se ingresa por primera vez a la máquina virtual y se procede a actualizarla ejecutando el comando "*apt-get update".*

- ***Instalación Jenkins en Ubuntu 20.04***

El integrador Jenkins se despliega en la máquina virtual "vm-secdevops-prod-eastus2-001" anteriormente desplegada. Como prerrequisito se debe instalar el JDK de Java y luego se procede a realizar la instalación de Jenkins siguiendo el paso a paso recomendado en [143] [144] como se observa en la Figura 4-5.

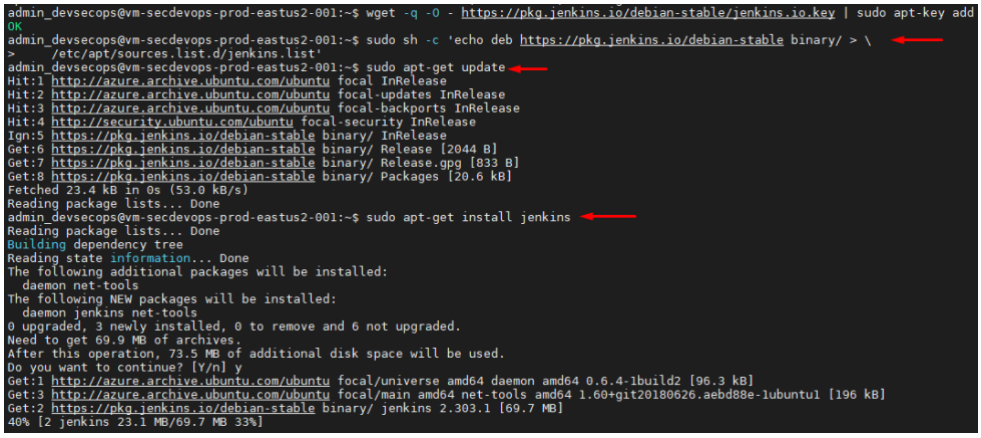



**Figura 4-5.** Comandos para instalar Jenkins en Ubuntu 20.04

Luego de finalizar el proceso de instalación se ejecuta el comando expuesto en la Figura 4-6 para comprobar el estado de Jenkins.

```
admin_devsecops@vm-secdevops-prod-eastus2-001:~$ sudo systemctl status jenkins
● jenkins.service - LSB: Start Jenkins at boot time
     Loaded: loaded (/etc/init.d/jenkins; generated)
     Active: active (exited) since Wed 2021-09-01 04:47:28 UTC; 29s ago
       Docs: man:systemd-sysv-generator(8)
      Tasks: 0 (limit: 4714)
     Memory: 0B
     CGroup: /system.slice/jenkins.service

Sep 01 04:47:27 vm-secdevops-prod-eastus2-001 systemd[1]: Starting LSB: Start Jenkins at boot time...
Sep 01 04:47:27 vm-secdevops-prod-eastus2-001 jenkins[4373]: Correct java version found
Sep 01 04:47:27 vm-secdevops-prod-eastus2-001 jenkins[4373]:  * Starting Jenkins Automation Server jenkins
Sep 01 04:47:27 vm-secdevops-prod-eastus2-001 su[4412]: (to jenkins) root on none
Sep 01 04:47:27 vm-secdevops-prod-eastus2-001 su[4412]: pam_unix(su-l:session): session opened for user jenkins by (uid=0)
Sep 01 04:47:27 vm-secdevops-prod-eastus2-001 su[4412]: pam_unix(su-l:session): session closed for user jenkins
Sep 01 04:47:28 vm-secdevops-prod-eastus2-001 jenkins[4373]:    ...done.
Sep 01 04:47:28 vm-secdevops-prod-eastus2-001 systemd[1]: Started LSB: Start Jenkins at boot time.
admin_devsecops@vm-secdevops-prod-eastus2-001:~$
```

**Figura 4-6.** Comprobación estado Jenkins

Para la configuración inicial, se accede mediante un navegador web con el DNS de la máquina virtual “vm-secdevops.eastus2.cloudapp.azure.com” y el puerto por defecto de Jenkins, 8080. En la Figura 4-7 y Figura 4-8 se muestra la ruta donde obtiene la clave temporal para continuar con la configuración y la clave temporal necesaria respectivamente.


**Figura 4-7.** Página inicial de Jenkins para su configuración

```
  GNU nano 4.8                                  /var/lib/jenkins/secrets/initialAdminPassword
49a91d8d2936487c82c6902bfefeb1d5
```

**Figura 4-8.** Clave temporal asignada para la configuración.

Se continua con la instalación de los complementos o plugins sugeridos por Jenkins, ver Figura 4-9.

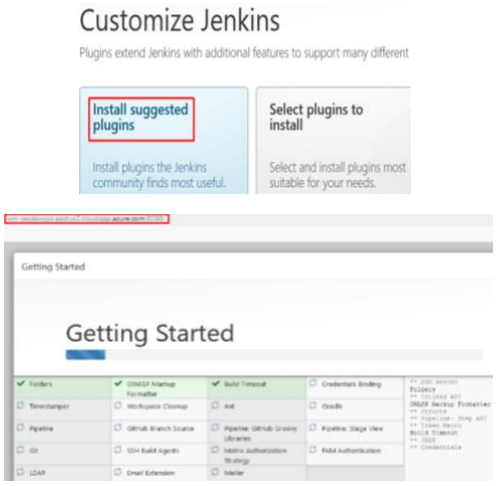



**Figura 4-9.** Instalación plugins sugeridos por Jenkins

Al finalizar la instalación de los plugins se debe crear el usuario administrador para la gestión de la herramienta, como se observa en la Figura 4-10.

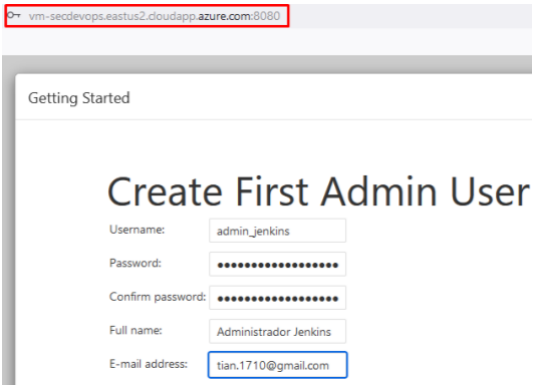



**Figura 4-10.** Creación del usuario administrador de Jenkins

En la Figura 4-11 se muestra la pantalla principal de Jenkins luego de la instalación y configuración del usuario administrador.

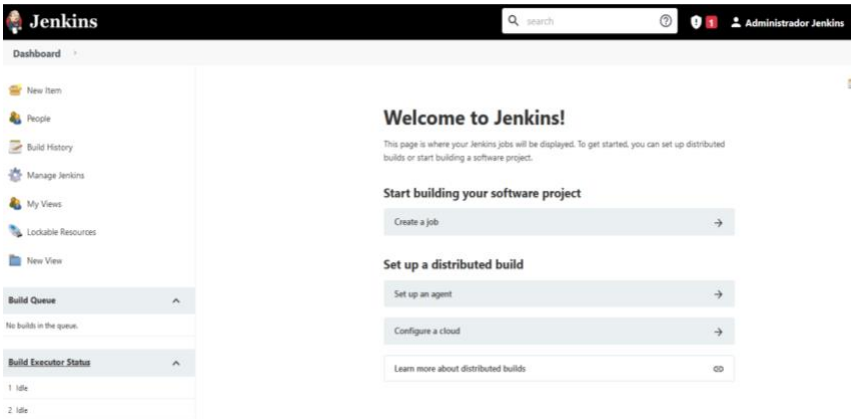



**Figura 4-11.** Pantalla principal Jenkins

- ***Instalación GitLab CE ubuntu 20.04***

Se continua con la instalación de la herramienta GitLab como repositorio centralizado de código. Para esto es necesario configurar otra máquina con característica similares a las de la máquina "vm-secdevops-prod-eastus2-001" desplegada anteriormente. Esta máquina nombrada "vm-gitlab-prod-eastus2-001" es desplegada en la subnet "snet-secdevops-prod-eastus2-002". Las características adicionales se pueden observar en la Figura 4-12.

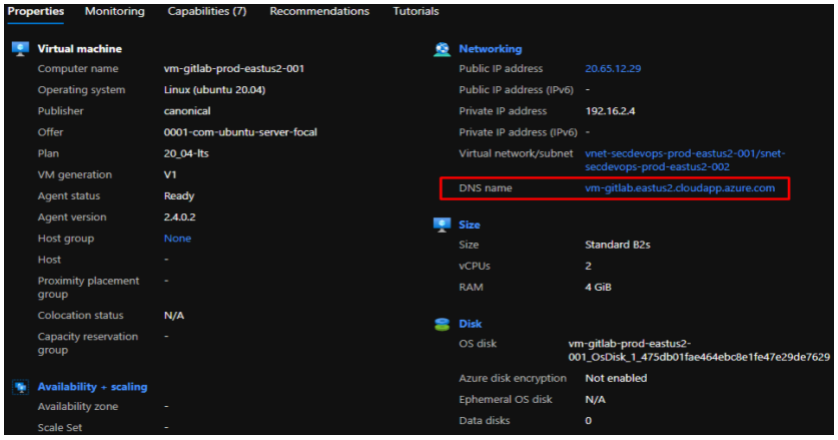



**Figura 4-12.** Características máquina virtual para la ejecución de GitLab

Basados en la documentación oficial [145] se procede a la instalación de GitLab iniciando con la instalación de las dependencias necesarias para su funcionamiento y luego con la descarga del paquete de GitLab del repositorio oficial como se observa en la Figura 4-13.

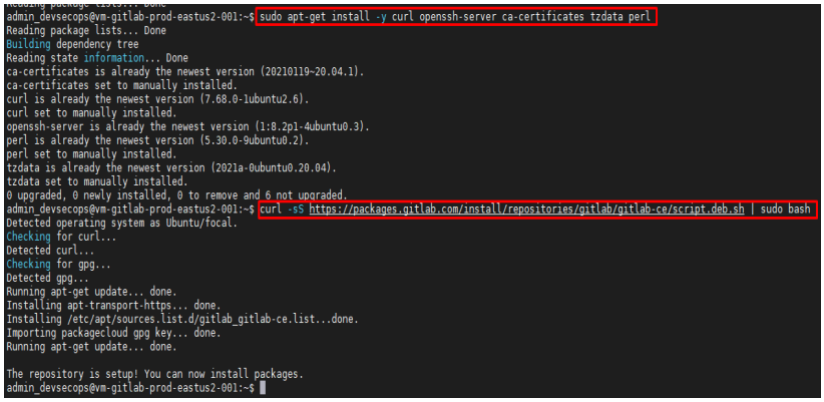



**Figura 4-13.** Descarga paquetes GitLab

Luego se procede a la instalación de los paquetes previamente instalados con el comando expuesto en la Figura 4-14.

```
admin_devsecops@vm-gitlab-prod-eastus2-001:~$ sudo apt-get install gitlab-ce
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  gitlab-ce
0 upgraded, 1 newly installed, 0 to remove and 6 not upgraded.
Need to get 942 MB of archives.
After this operation, 2546 MB of additional disk space will be used.
Get:1 https://packages.gitlab.com/gitlab/gitlab-ce/ubuntu focal/main amd64 gitlab-ce amd64 14.2.3-ce.0 [942 MB]
Fetched 942 MB in 19s (50.2 MB/s)
Selecting previously unselected package gitlab-ce.
(Reading database ... 59501 files and directories currently installed.)
Preparing to unpack .../gitlab-ce_14.2.3-ce.0_amd64.deb ...
Unpacking gitlab-ce (14.2.3-ce.0) ...
```

```
Thank you for installing GitLab!
GitLab was unable to detect a valid hostname for your instance.
Please configure a URL for your GitLab instance by setting `external_url`
configuration in /etc/gitlab/gitlab.rb file.
Then, you can start your GitLab instance by running the following command:
  sudo gitlab-ctl reconfigure

For a comprehensive list of configuration options please see the Omnibus GitLab readme
https://gitlab.com/gitlab-org/omnibus-gitlab/blob/master/README.md

Help us improve the installation experience, let us know how we did with a 1 minute survey:
https://gitlab.fra1.qualtrics.com/jfe/form/SV_6kVqZANThUQ1bZb?installation=omnibus&release=14-2
```

**Figura 4-14.** Ejecución comando para la instalación de GitLab

En la Figura 4-15 se muestra la ejecución del comando "*gitlab-ctl reconfigure*" para finalizar la instalación.

```
admin_devsecops@vm-gitlab-prod-eastus2-001:~$ sudo gitlab-ctl reconfigure
Starting Chef Infra Client, version 15.14.0
resolving cookbooks for run list: ["gitlab"]
Synchronizing Cookbooks:
  - gitlab (0.0.1)
  - package (0.1.0)
  - logrotate (0.1.0)
  - postgresql (0.1.0)
  - redis (0.1.0)
  - registry (0.1.0)
  - monitoring (0.1.0)
  - mattermost (0.1.0)
  - consul (0.1.0)
  - gitlab-kas (0.1.0)
  - gitlab-pages (0.1.0)
  - praefect (0.1.0)
  - gitaly (0.1.0)
  - runit (5.1.3)
  - letsencrypt (0.1.0)
  - nginx (0.1.0)
  - acme (4.1.3)
```

**Figura 4-15**. Finalización de la instalación de GitLab

Al igual que la configuración de Jenkins, por defecto se crea el usuario "root" con una clave temporal que se usará luego para acceder por primera vez, como se muestra en la Figura 4-16.

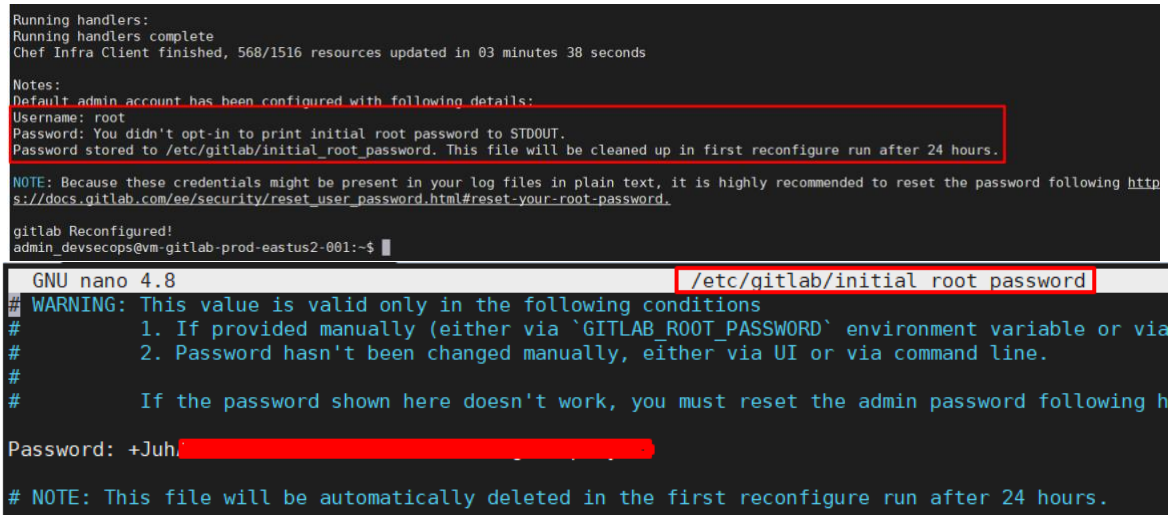



**Figura 4-16.** Ruta para obtener la clave temporal del usuario root

Se inicia el servicio de GitLab con el comando *"sudo gitlab-ctl start".* Al tener arriba el servicio se procede a acceder mediante un navegador web para visualizar la pantalla inicial de administración y los componentes descargados. Esto se puede observar en las figuras: Figura 4-17, Figura 4-18, Figura 4-19 y Figura 4-20 respectivamente.

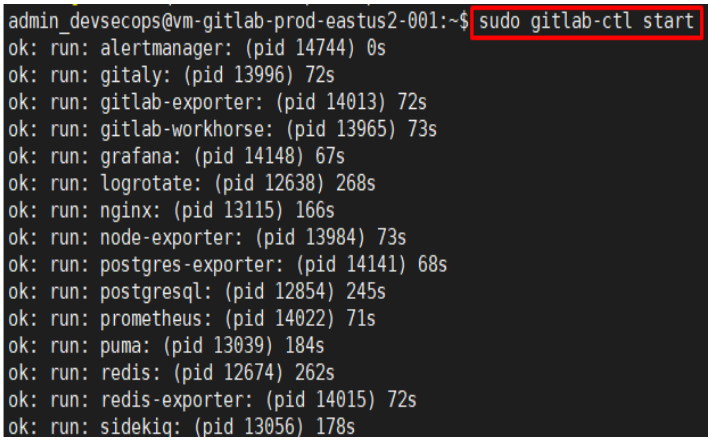



**Figura 4-17**. Ejecución comando para iniciar GitLab

Se accede mediante el navegador con el DNS "vm-gitlab-prod-eastus2-001" de la máquina creada anteriormente.

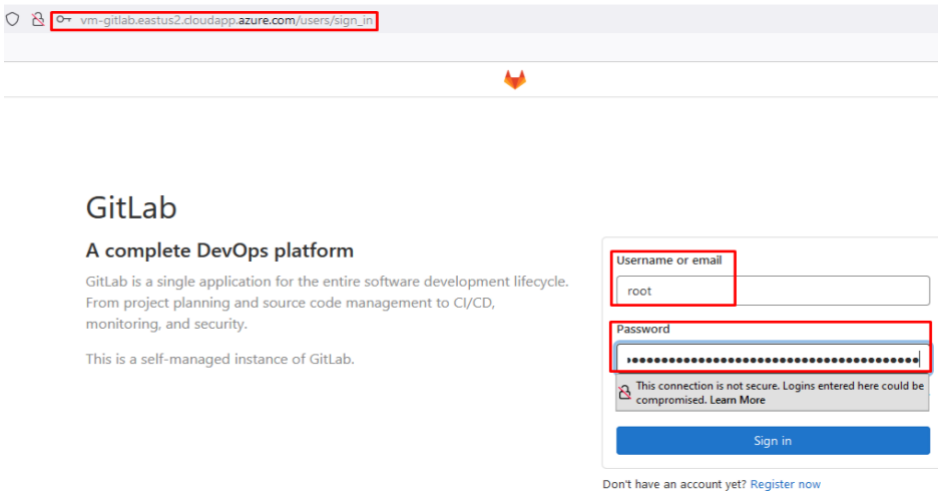



**Figura 4-18.** Acceso a GitLab

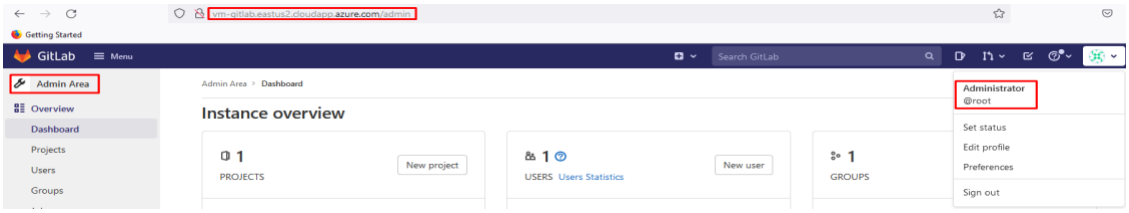



**Figura 4-19.** Pantalla de administración de GitLab

Componentes y versiones instalados.

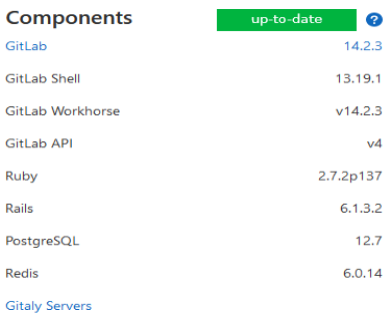



**Figura 4-20.** Componentes instalados con sus respectivas versiones

Como buena práctica de seguridad se procede a cambiar la contraseña temporal asignada al usuario root y realizar una prueba inicial de integración con el IDE VsCode.

- ***Instalación SonarQube 9.0.1 Ubuntu 20.04***

La instalación de la herramienta está guiada por la documentación oficial [146] y otras fuentes [147] [148].La configuración y el despliegue de SonarQube se ejecuta en la máquina virtual "*vm-secdevops-prod-eastus2-001"*, máquina virtual dónde se desplegó Jenkins. Iniciamos validando la versión del JDK de Java instalado en la máquina virtual.

```
admin_devsecops@vm-secdevops-prod-eastus2-001:~$ java --version
openjdk 11.0.11 2021-04-20
OpenJDK Runtime Environment (build 11.0.11+9-Ubuntu-0ubuntu2.20.04)
OpenJDK 64-Bit Server VM (build 11.0.11+9-Ubuntu-0ubuntu2.20.04, mixed mode)
```

**Figura 4-21.** Versión JDK del Java instalado

Se continua con la configuración de las siguientes variables con su respectivo valor, tener en cuenta que para se guarden los cambios es necesario reiniciar la máquina virtual, las variables se muestran en la Figura 4-22.

```
admin_devsecops@vm-secdevops-prod-eastus2-001:~$ sudo sysctl -w vm.max_map_count=524288
vm.max_map_count = 524288
admin_devsecops@vm-secdevops-prod-eastus2-001:~$ sudo sysctl -w fs.file-max=131072
fs.file-max = 131072
admin_devsecops@vm-secdevops-prod-eastus2-001:~$ ulimit -n 131072
admin_devsecops@vm-secdevops-prod-eastus2-001:~$ ulimit -u 8192
```

**Figura 4-22.** Configuración de las variables para el despliegue de SonarQube

Adicional es necesario modificar los límites del sistema kernel como se visualiza en la Figura 4-23.

```
  GNU nano 4.8                                  /etc/sysctl.conf
#
# Do not accept ICMP redirects (prevent MITM attacks)
#net.ipv4.conf.all.accept_redirects = 0
#net.ipv6.conf.all.accept_redirects = 0
# _or_
# Accept ICMP redirects only for gateways listed in our default
# gateway list (enabled by default)
# net.ipv4.conf.all.secure_redirects = 1
#
# Do not send ICMP redirects (we are not a router)
#net.ipv4.conf.all.send_redirects = 0
#
# Do not accept IP source route packets (we are not a router)
#net.ipv4.conf.all.accept_source_route = 0
#net.ipv6.conf.all.accept_source_route = 0
#
# Log Martian Packets
#net.ipv4.conf.all.log_martians = 1
#

###################################################################
# Magic system request Key
# 0=disable, 1=enable all, >1 bitmask of sysrq functions
# See https://www.kernel.org/doc/html/latest/admin-guide/sysrq.html
# for what other values do
#kernel.sysrq=438

vm.max_map_count=262144
fs.file-max=65536
ulimit -n 65536
ulimit -u 4096
```

**Figura 4-23.** Modificación de los límites del kernel de la vm

Para el consumo del componente de sonar es necesario la creación de un usuario dedicado por lo que en la Figura 4-24 se muestra la creación del usuario “sonar” con su respectiva contraseña.

```
admin_devsecops@vm-secdevops-prod-eastus2-001:~$ sudo useradd sonar
useradd: user 'sonar' already exists
admin_devsecops@vm-secdevops-prod-eastus2-001:~$ sudo passwd sonar
New password:
Retype new password:
passwd: password updated successfully
admin_devsecops@vm-secdevops-prod-eastus2-001:~$
```

**Figura 4-24**. Creación del usuario sonar con su respectiva contraseña

Para el correcto funcionamiento de la herramienta SonarQube se debe desplegar un motor de base de datos PostgreSQL, dado esto, se expone a continuación el procedimiento para instalar una base de datos PostgreSQL versión 13. Se inicia con la descarga de los paquetes necesarios desde la fuente oficial y la instalación de estos, como se observa en la Figura 4-25.

```
admin_devsecops@vm-secdevops-prod-eastus2-001:~$ echo "deb [arch=amd64] http://apt.postgresql.org/pub/repos/apt/ focal-pgdg main" | sudo tee
/etc/apt/sources.list.d/postgresql.list
deb [arch=amd64] http://apt.postgresql.org/pub/repos/apt/ focal-pgdg main
admin_devsecops@vm-secdevops-prod-eastus2-001:~$ sudo apt update
Hit:1 http://azure.archive.ubuntu.com/ubuntu focal InRelease
Hit:2 http://azure.archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:3 http://azure.archive.ubuntu.com/ubuntu focal-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu focal-security InRelease
Ign:5 https://pkg.jenkins.io/debian-stable binary/ InRelease
Hit:6 https://pkg.jenkins.io/debian-stable binary/ Release
Get:8 http://apt.postgresql.org/pub/repos/apt focal-pgdg InRelease [86.7 kB]
Get:9 http://apt.postgresql.org/pub/repos/apt focal-pgdg/main amd64 Packages [213 kB]
Fetched 300 kB in 1s (486 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
6 packages can be upgraded. Run 'apt list --upgradable' to see them.
admin_devsecops@vm-secdevops-prod-eastus2-001:~$
```

**Figura 4-25**. Descarga e instalación de los paquetes de PostreSQL

Se ejecutan los comandos expuestos en la Figura 4-26 para la instalación de la base de datos PostgreSQL versión 13 y la validación de esta.

```
admin_devsecops@vm-secdevops-prod-eastus2-001:~$ sudo apt install postgresql-13
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libcommon-sense-perl libjson-perl libjson-xs-perl libllvm9 libpq5 libtypes-serialiser-perl pgdg-keyring postgresql-client-13
  postgresql-client-common postgresql-common sysstat
Suggested packages:
  postgresql-doc-13 isag
The following NEW packages will be installed:
  libcommon-sense-perl libjson-perl libjson-xs-perl libllvm9 libpq5 libtypes-serialiser-perl pgdg-keyring postgresql-13
  postgresql-client-13 postgresql-client-common postgresql-common sysstat
0 upgraded, 12 newly installed, 0 to remove and 6 not upgraded.
Need to get 32.6 MB of archives.
After this operation, 129 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://azure.archive.ubuntu.com/ubuntu focal/main amd64 libcommon-sense-perl amd64 3.74-2build6 [20.1 kB]
Get:2 http://azure.archive.ubuntu.com/ubuntu focal/main amd64 libjson-perl all 4.02000-2 [80.9 kB]
Get:3 http://azure.archive.ubuntu.com/ubuntu focal/main amd64 libtypes-serialiser-perl all 1.0-1 [12.1 kB]
Get:4 http://azure.archive.ubuntu.com/ubuntu focal/main amd64 libjson-xs-perl amd64 4.020-1build1 [83.7 kB]
Get:5 http://azure.archive.ubuntu.com/ubuntu focal/main amd64 libllvm9 amd64 1:9.0.1-12 [14.8 MB]
Get:6 http://azure.archive.ubuntu.com/ubuntu focal-updates/main amd64 sysstat amd64 12.2.0-2ubuntu0.1 [448 kB]
Get:7 http://apt.postgresql.org/pub/repos/apt focal-pgdg/main amd64 libpq5 amd64 13.4-1.pgdg20.04+1 [179 kB]
admin_devsecops@vm-secdevops-prod-eastus2-001:~$ sudo systemctl status postgresql
● postgresql.service - PostgreSQL RDBMS
     Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; vendor preset: enabled)
     Active: active (exited) since Thu 2021-09-02 05:11:58 UTC; 55s ago
   Main PID: 4861 (code=exited, status=0/SUCCESS)
      Tasks: 0 (limit: 9545)
     Memory: 0B
     CGroup: /system.slice/postgresql.service

Sep 02 05:11:58 vm-secdevops-prod-eastus2-001 systemd[1]: Starting PostgreSQL RDBMS...
Sep 02 05:11:58 vm-secdevops-prod-eastus2-001 systemd[1]: Finished PostgreSQL RDBMS.
admin_devsecops@vm-secdevops-prod-eastus2-001:~$
```

**Figura 4-26.** Instalación y verificación de PostgreSQL

Para la base de datos es necesario crear un usuario dedicado, con el fin de gestionar esta.

```
admin_devsecops@vm-secdevops-prod-eastus2-001:~$ sudo passwd postgres
New password:
Retype new password:
passwd: password updated successfully
```

**Figura 4-27.** Creación del usuario postgres con su respectiva contraseña.

Validación del usuario y cambio al usuario “postgres”, ver Figura 4-28.

```
admin_devsecops@vm-secdevops-prod-eastus2-001:~$ su - postgres
Password:
postgres@vm-secdevops-prod-eastus2-001:~$
```

**Figura 4-28.** Validación del usuario postgres

Al estar logueado como usuario “postgres” se crea el usuario “sonaruser” para acceder y gestionar la base de datos que usará SonarQube, el comando para la creación del usuario “sonaruser” se puede visualizar en la Figura 4-29.

```
postgres@vm-secdevops-prod-eastus2-001:~$ createuser sonaruser
```

**Figura 4-29.** Creación del usuario “sonaruser”

Se crea la base de datos “sonardb” con los privilegios necesarios para el usuario “sonaruser” para la gestión de esta, como se expone en la Figura 4-30 y Figura 4-31.

```
postgres@vm-secdevops-prod-eastus2-001:~$ psql
psql (13.4 (Ubuntu 13.4-1.pgdg20.04+1))
Type "help" for help.

postgres=# ALTER USER sonaruser WITH ENCRYPTED password 'Y[redacted];
ALTER ROLE
postgres=# CREATE DATABASE sonardb OWNER sonaruser;
CREATE DATABASE
postgres=# GRANT ALL PRIVILEGES ON DATABASE sonardb to sonaruser;
GRANT
postgres=# \q+
```

**Figura 4-30.** Creación base de datos “sonardb”

En este punto se finaliza con la configuración de los prerrequisitos requeridos. A continuación, se procede a la instalación de SonarQube iniciando con la descarga y la tarea de descomprimir el comprimido en el directorio “/otp”, así como se observa en la Figura 4-31.

```
admin_devsecops@vm-secdevops-prod-eastus2-001:~$ wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.0.1.46107.zip
--2021-09-02 05:19:23--  https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.0.1.46107.zip
Resolving binaries.sonarsource.com (binaries.sonarsource.com)... 91.134.125.245
Connecting to binaries.sonarsource.com (binaries.sonarsource.com)|91.134.125.245|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 290201762 (277M) [application/zip]
Saving to: ‘sonarqube-9.0.1.46107.zip’

sonarqube-9.0.1.46107.zip          100%[===================================================================>] 276.76M  31.1MB/s    in 9.6s

2021-09-02 05:19:33 (28.9 MB/s) - ‘sonarqube-9.0.1.46107.zip’ saved [290201762/290201762]

admin_devsecops@vm-secdevops-prod-eastus2-001:~$ sudo unzip sonarqube-*.zip -d /opt
Archive:  sonarqube-9.0.1.46107.zip
   creating: /opt/sonarqube-9.0.1.46107/
   creating: /opt/sonarqube-9.0.1.46107/bin/
   creating: /opt/sonarqube-9.0.1.46107/bin/jsw-license/
  inflating: /opt/sonarqube-9.0.1.46107/bin/jsw-license/LICENSE.txt
   creating: /opt/sonarqube-9.0.1.46107/bin/windows-x86-64/
  inflating: /opt/sonarqube-9.0.1.46107/bin/windows-x86-64/StopNTService.bat
   creating: /opt/sonarqube-9.0.1.46107/bin/windows-x86-64/lib/
  inflating: /opt/sonarqube-9.0.1.46107/bin/windows-x86-64/lib/wrapper.dll
```

**Figura 4-31.** Descarga y descomprimir los binarios

Luego de tener los binarios en la ruta “/opt” es necesario moverlos a la ruta “/opt/sonarqube” y garantizar el acceso al usuario “sonar” a esta ruta. Los comandos para realizar estas acciones se muestran en la Figura 4-32.

```
admin_devsecops@vm-secdevops-prod-eastus2-001:~$ sudo mv /opt/sonarqube-* /opt/sonarqube
admin_devsecops@vm-secdevops-prod-eastus2-001:~$ sudo chown sonar:sonar /opt/sonarqube -R
```

**Figura 4-32.** Movimiento de los binarios y acceso garantizado al usuario sonar.

En el archivo “sonar.properties” ubicado en la ruta “/opt/sonarqube/conf” se proceden a configurar los parámetros para el acceso a la base de datos: Endpoint donde está corriendo la base de datos, base de datos “sonardb”, y el usuario y la contraseña, como se puede observar en la Figura 4-33.

```
  GNU nano 4.8                                  /opt/sonarqube/conf/sonar.properties
#   upper-cased name of the property where all the dot ('.') and dash ('-') characters are replaced by
#   underscores ('_'). For example, to override 'sonar.web.systemPasscode' use 'SONAR_WEB_SYSTEMPASSCODE'.
# - be encrypted. See https://redirect.sonarsource.com/doc/settings-encryption.html

#--------------------------------------------------------------------------------------------------
# DATABASE
#
# IMPORTANT:
# - The embedded H2 database is used by default. It is recommended for tests but not for
#   production use. Supported databases are Oracle, PostgreSQL and Microsoft SQLServer.
# - Changes to database connection URL (sonar.jdbc.url) can affect SonarSource licensed products.

# User credentials.
# Permissions to create tables, indices and triggers must be granted to JDBC user.
# The schema must be created first.
sonar.jdbc.username=sonaruser
Sonar.jdbc.password=Y[redacted]

#sonar.jdbc.url=jdbc:oracle:thin:@localhost:1521/XE


#-----PostgreSQL 9.6 or greater
# By default the schema named "public" is used. It can be overridden with the parameter "currentSchema"
sonar.jdbc.url=jdbc:postgresql://localhost/sonardb
```

**Figura 4-33**. Configuración cadena de conexión a la base de datos “sonardb”

Adicionalmente, es importante realizar las siguientes configuraciones para finalizar con la instalación de la herramienta. En el ejecutable “sonar.sh” se debe realizar la siguiente configuración, como se observa en la Figura 4-34.

```
  GNU nano 4.8                                  /opt/sonarqube/bin/linux-x86-64/sonar.sh
#  the PID file and wrapper.log files.  Failure to be able to write the log
#  file will cause the Wrapper to exit without any way to write out an error
#  message.
# NOTE - This will set the user which is used to run the Wrapper as well as
#  the JVM and is not useful in situations where a privileged resource or
#  port needs to be allocated prior to the user being changed.
RUN_AS_USER=sonar

# The following two lines are used by the chkconfig command. Change as is
#  appropriate for your application.  They should remain commented.
```

**Figura 4-34.** Configuración en el ejecutable sonar.sh

Por último, es necesario crear el archivo de servicio “sonar.service” para iniciar y parar el servicio de SonarQube, Figura 4-35.

```
 GNU nano 4.8                                  /etc/systemd/system/sonar.service
[Unit]
Description=SonarQube service
After=syslog.target network.target

[Service]
Type=forking

ExecStart=/opt/sonarqube/bin/linux-x86-64/sonar.sh start
ExecStop=/opt/sonarqube/bin/linux-x86-64/sonar.sh stop

User=sonar
Group=sonar
Restart=always

LimitNOFILE=65536
LimitNPROC=4096

[Install]
WantedBy=multi-user.target
```

**Figura 4-35.** Creación archivo “sonar.service”

Se ha realizado paso a paso la instalación de la herramienta SonarQube, para comprobar el funcionamiento, se inicia el servicio “sonar” y se ingresa mediante el navegador web a la URL: “http: vm-secdevops-prod-eastus2-001:9000” con las credenciales por defecto admin/admin. En la Figura 4-36 se visualiza el resultado del inicio del servicio.

```
root@vm-secdevops-prod-eastus2-001:/opt/sonarqube/conf# sudo systemctl enable sonar
root@vm-secdevops-prod-eastus2-001:/opt/sonarqube/conf# sudo systemctl start sonar
root@vm-secdevops-prod-eastus2-001:/opt/sonarqube/conf# sudo systemctl status sonar
● sonar.service - SonarQube service
     Loaded: loaded (/etc/systemd/system/sonar.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2021-09-02 06:23:06 UTC; 10min ago
   Main PID: 10985 (wrapper)
      Tasks: 153 (limit: 9545)
     Memory: 1.7G
     CGroup: /system.slice/sonar.service
             ├─10985 /opt/sonarqube/bin/linux-x86-64/./wrapper /opt/sonarqube/bin/linux-x86-64/../../conf/wrapper.conf wrapper.syslog.ident=>
             ├─10987 /usr/lib/jvm/java-11-openjdk-amd64/bin/java -Dsonar.wrapped=true -Djava.awt.headless=true -Xms8m -Xmx32m -Djava.library>
             ├─11015 /usr/lib/jvm/java-11-openjdk-amd64/bin/java -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=75 -XX:+UseCMSIn>
             ├─11100 /usr/lib/jvm/java-11-openjdk-amd64/bin/java -Djava.awt.headless=true -Dfile.encoding=UTF-8 -Djava.io.tmpdir=/opt/sonarq>
             └─11192 /usr/lib/jvm/java-11-openjdk-amd64/bin/java -Djava.awt.headless=true -Dfile.encoding=UTF-8 -Djava.io.tmpdir=/opt/sonarq>

Sep 02 06:23:05 vm-secdevops-prod-eastus2-001 systemd[1]: Starting SonarQube service...
Sep 02 06:23:05 vm-secdevops-prod-eastus2-001 sonar.sh[10936]: Starting SonarQube...
Sep 02 06:23:06 vm-secdevops-prod-eastus2-001 sonar.sh[10936]: Started SonarQube.
Sep 02 06:23:06 vm-secdevops-prod-eastus2-001 systemd[1]: Started SonarQube service.
```

**Figura 4-36**. Inicio servicio sonar y verificación del estado

En la Figura 4-37, se observa que la herramienta se está ejecutando correctamente. Importante, cambiar la contraseña luego de iniciar la sesión por primera vez.

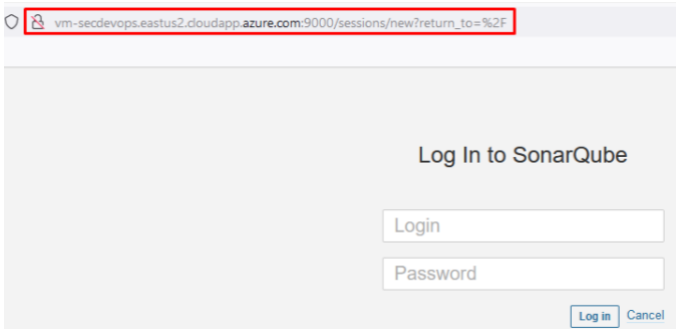



**Figura 4-37.** Consumo de la herramienta por la WEB

Es importante mencionar que al instalar la herramienta en la máquina virtual donde se está ejecutando Jenkins es necesario monitorear el consumo tanto de vCPU como de memoria RAM

para así determinar el tamaño ideal para trabajar. En el caso nuestro, el tamaño elegido para la máquina es de 2vCPU y 8GB de RAM.

- ***Instalación Gradle y JUnit***

Haciendo uso de la opción de configuración global de herramientas en Jenkins, se procede a la instalación de la herramienta Gradle. En la pestaña *Manage Jenkins –> Global Tool Configuration* se configura la opción de instalar automáticamente la versión 5.6, como se observa en la Figura 4-38.

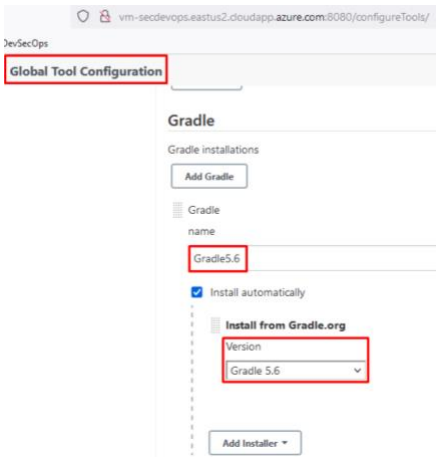



**Figura 4-38**. Configuración instalación Gradle

Desde Jenkins es necesario instalar los plugins de Gradle y JUnit para el correcto funcionamiento de los componentes propuestos, ver Figura 4-39 y Figura 4-40 respectivamente.

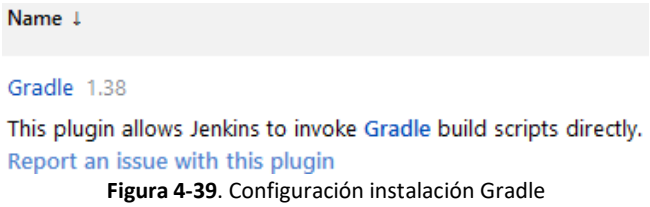



**Figura 4-39**. Configuración instalación Gradle

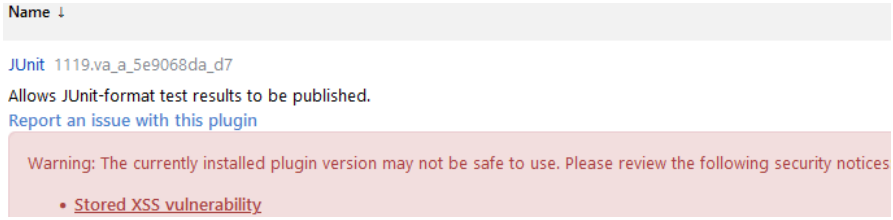



**Figura 4-40.** Configuración instalación Gradle

- ***Instalación Docker***

A continuación, se expone el paso a paso ejecutado para la instalación de Docker en la máquina "vm-secdevops-prod-eastus2-001". Es necesario mencionar que los comandos que se ejecutan a continuación se deben ejecutar con el usuario "root". Basado en esto, se inicia con la descarga de los paquetes necesarios [149], como se observa en la Figura 4-41.

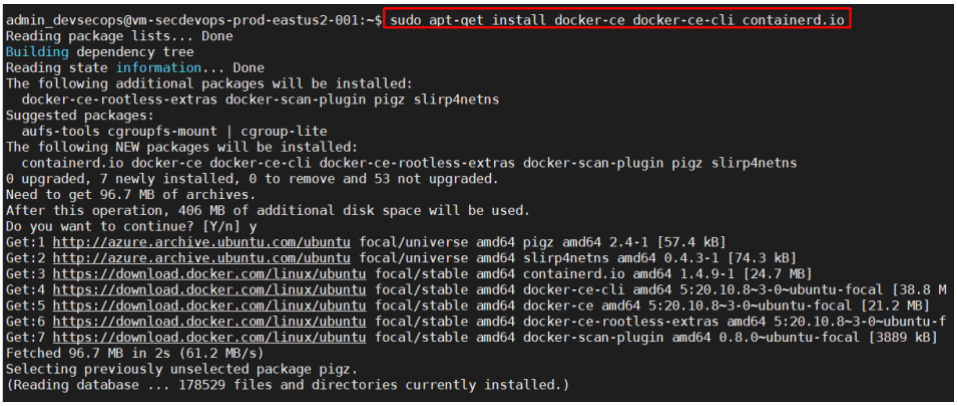



**Figura 4-41.** Descarga paquetes Docker

Se validan las versiones correspondientes a los paquetes descargador y se procede a la instalación de la versión seleccionada, como se observan en las Figura 4-42 y Figura 4-43.

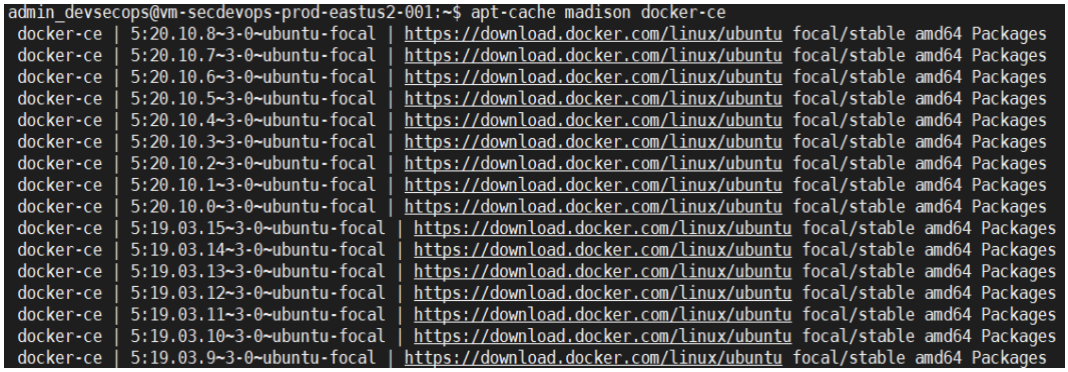



**Figura 4-42.** Visualización versiones descargadas

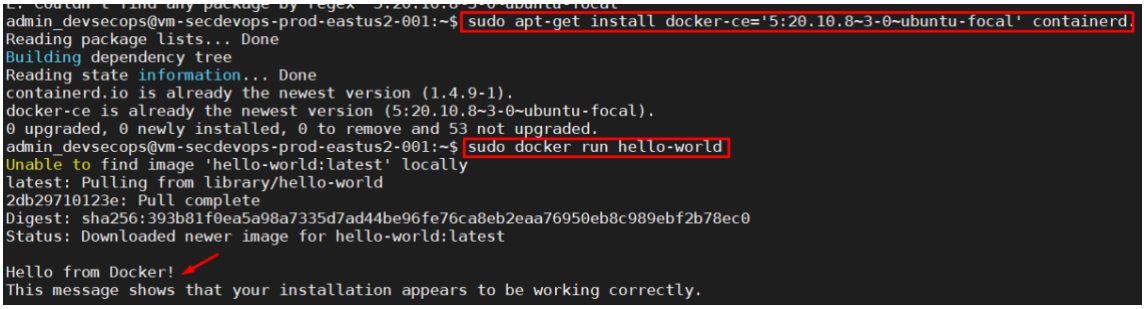



**Figura 4-43.** Instalación de Docker

En Jenkins es necesario instalar los plugins que se muestran en la siguiente Figura 4-44.

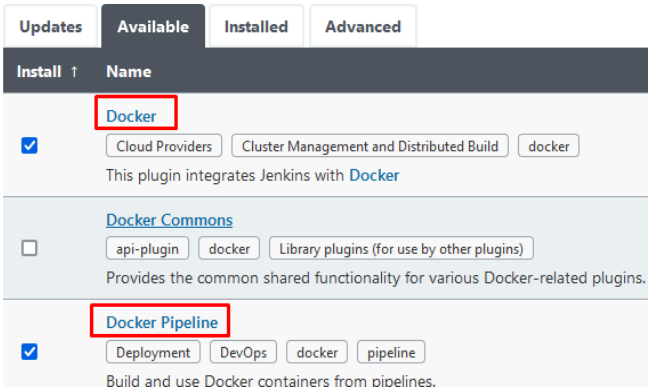



**Figura 4-44.** Instalación plugins Docker en Jenkins

- ***Configuración ACR***

Se realiza la creación del contenedor de registros con la configuración básica que ofrece Azure, en la Figura 4-45 muestran los parámetros de configuración que se pueden establecer

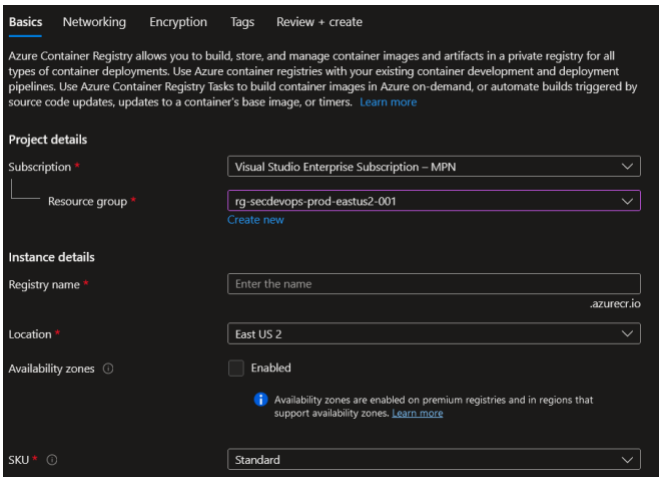



**Figura 4-45**. Parámetros de configuración del ACR

En la Figura 4-46 y Figura 4-47 se muestra el resumen de las características del ACR creado y las llaves de acceso a este.

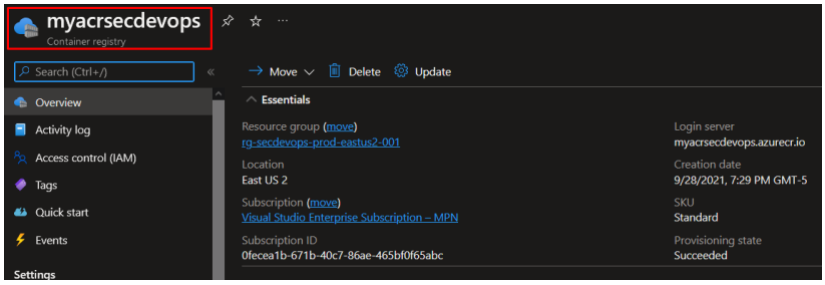



**Figura 4-46**. Resumen del ACR creado

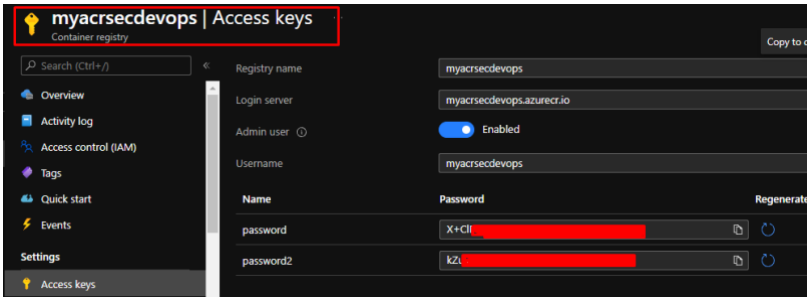



**Figura 4-47.** Llaves de acceso del ACR

A partir de las llaves de acceso generadas en Azure se crea la credencial en Jenkins, como se observa en la Figura 4-48.

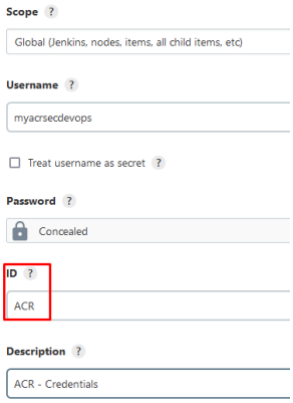



**Figura 4-48.** Creación credencial para el ACR

- ***Configuración Kubernetes en Azure (AKS)***

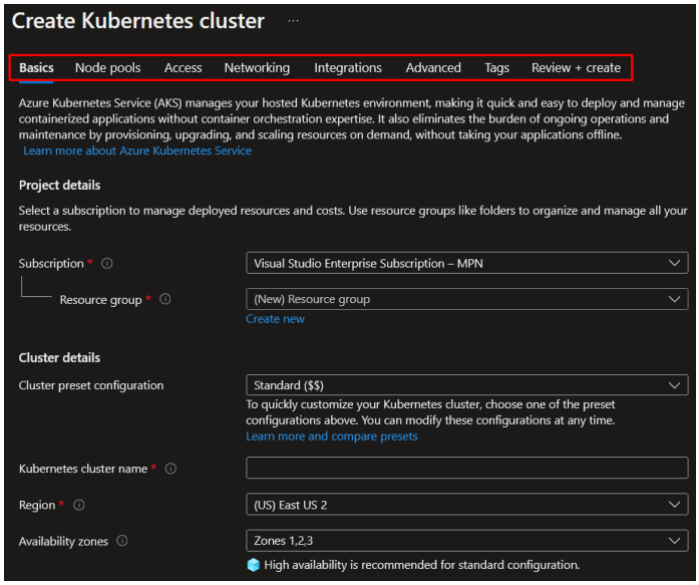



**Figura 4-49.** Propiedades a configurar en el clúster de kubernetes – Parte 1

Para la creación del clúster de kubernetes se hace uno del servicio administrado de Azure. A continuación, se presenta el paso a paso seguido para la configuración básica del clúster. En la Figura 4-49 se muestra las categorías de las propiedades a configurar en el clúster de kubernetes, entre ellas: Las zonas de disponibilidad, la región donde se va a desplegar el nodo principal.

También es necesario definir el número de nodos y el tamaño de este, como se observa en la Figura 4-50.

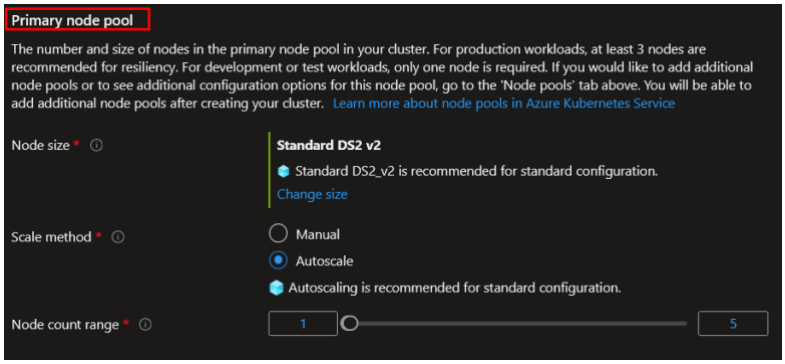



**Figura 4-50**. Propiedades a configurar en el clúster de kubernetes – Parte 2

A nivel de redes en la Figura 4-51 se observan los parámetros que se pueden configurar.

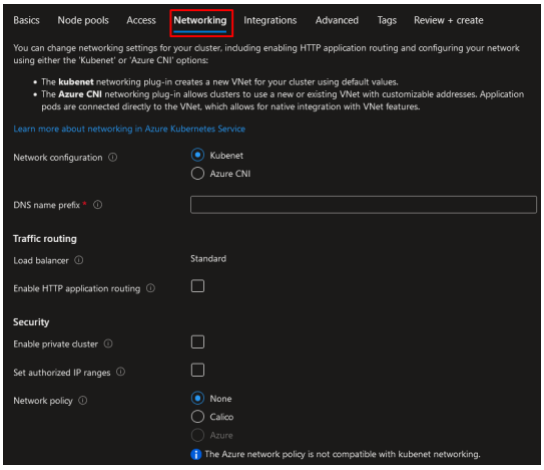



**Figura 4-51.** Propiedades a configurar en el clúster de kubernetes – Parte 3

Luego de la configuración básica del clúster, se muestra a continuación en la Figura 4-52 el resumen de las propiedades del clúster creado.

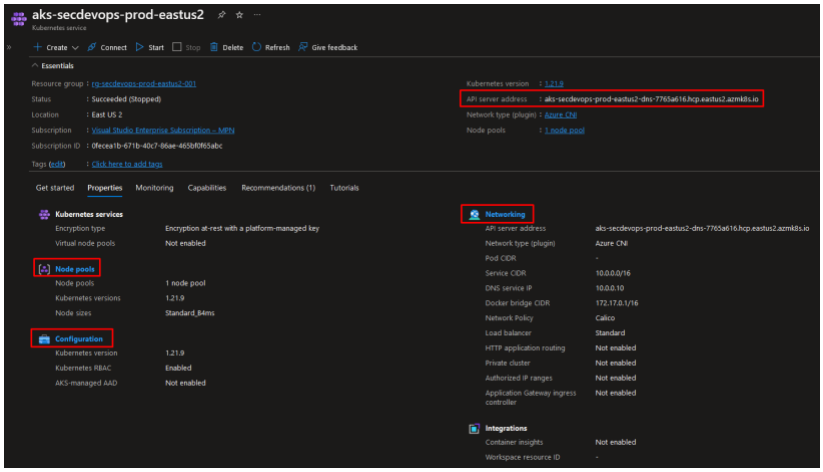



**Figura 4-52.** Resumen propiedades del clúster configurado

Para la descarga del archivo “config”, el cual se usará para la conexión al clúster de kubernetes se sigue el procedimiento expuesto a continuación. En la consola del clúster de kubernetes creado se accede a la opción “Connect” la cual indica el procedimiento a seguir, como se observa en la Figura 4-53.

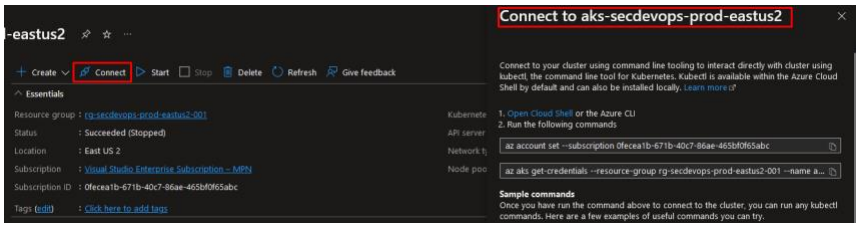



**Figura 4-53.** Procedimiento para conectar al clúster de kubernetes creado

Al ejecutar los comandos necesarios, se descarga el archivo “config” en la ruta que se muestra en la Figura 4-54.

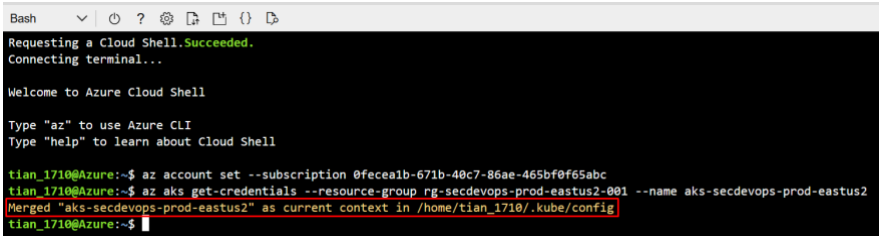



**Figura 4-54.** Ruta donde se descarga el archivo “config**”**

En la Figura 4-55 se observa que efectivamente el archivo fue descargado exitosamente.

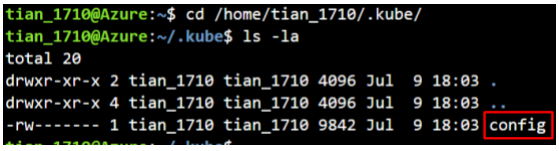



**Figura 4-55.** Validación de la descarga del archivo “config”

Y finalmente se procede a descargar el archivo localmente, como se observa en la Figura 4-56 para luego usarlo en la máquina donde se va a administrar el clúster de kubernetes.

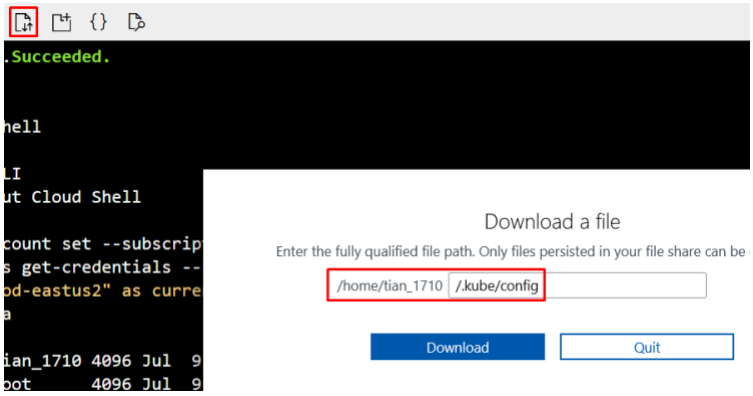



**Figura 4-56.** Procedimiento para descargar el archivo "config" localmente

Para la gestión del clúster en la máquina virtual "vm-secdevops-prod-eastus2-001" es necesario configurar el cliente de kubernetes "kubectl", como se muestra en la Figura 4-57.

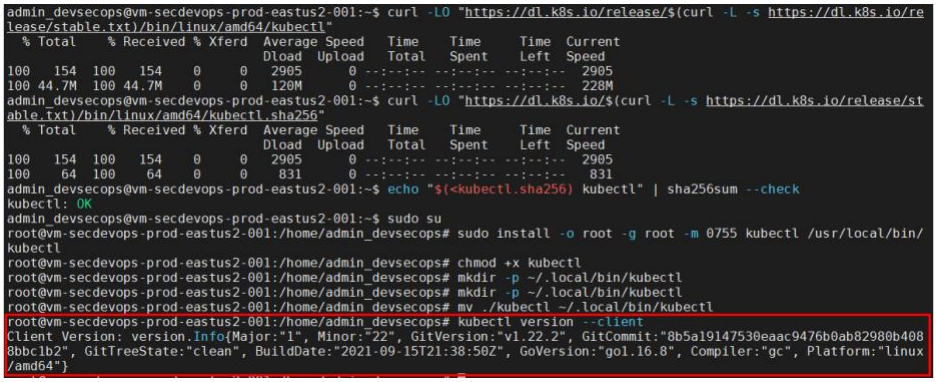



**Figura 4-57**. Configuración cliente kubectl

En este punto ha finalizado el despliegue del clúster de kubernetes, por lo que a continuación se muestran los pasos adicionales que se deben realizar en Jenkins para la integración. Iniciamos con la instalación de los plugins expuestos en la Figura 4-58.


**Figura 4-58**. Plugins instalados en Jenkins

Y finalizamos con la creación de la credencial para la conexión al clúster de kubernetes tipo “Secret File” haciendo uso del archivo de configuración “config” previamente descargado. La credencial “AKS-Config” configurada se muestra en la Figura 4-59.


**Figura 4-59**. Credencial para la conexión al clúster de kubernetes configurada

**Integración de las herramientas SonarQube y GitLab con Jenkins**

*Integración Jenkins-SonarQube.*

La referencia [150] sirve de guía para la integración expuesta posteriormente. Se inicia con la configuración en SonarQube. En la pestaña: *myaccount->security*, se genera un token para la conexión con Jenkins, como se observa en la Figura 4-60.

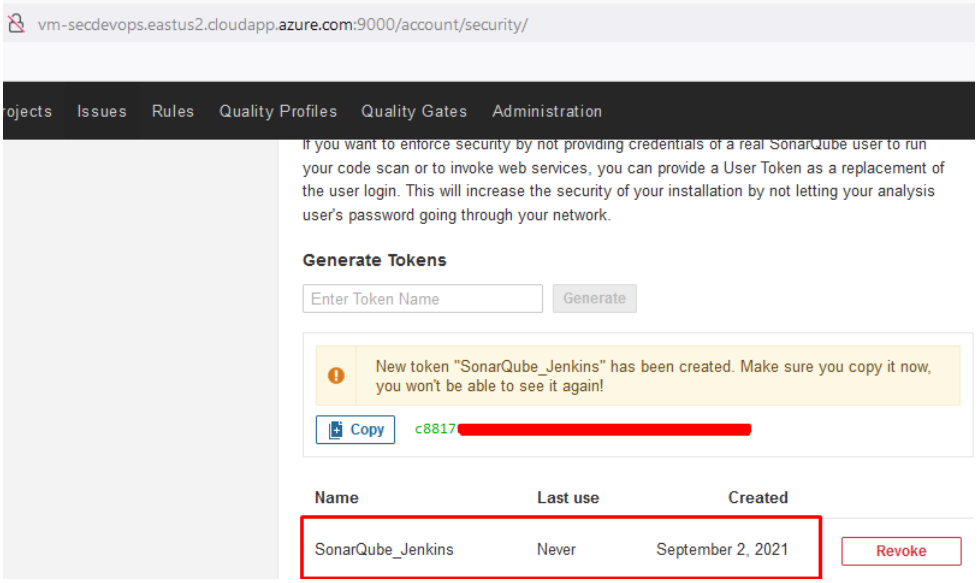



**Figura 4-60.** Generación token en SonarQube

A continuación, se exponen las configuraciones que se deben llevar acabo en Jenkins. En la pestaña de Jenkins “*Manage Jenkins*”, se procede a la instalación del plugin *“SonarQube Scanner”*, como se observa en la Figura 4-61.

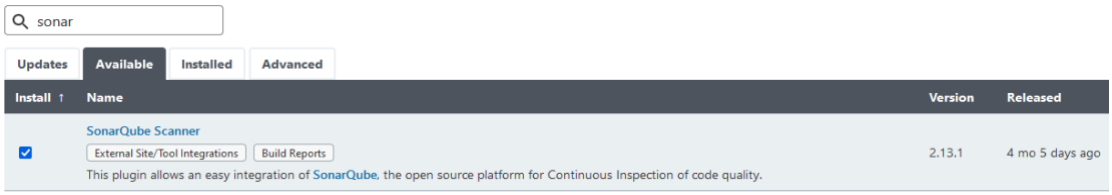



**Figura 4-61.** Generación token en SonarQube

Para la configuración accedemos a la pestaña "*Manage Jenkins→ Global Tool Configuration"* de Jenkins. En esta sección se configura la ruta donde se ejecutará el sonar scanner "*opt/sonarqube*" (Ruta configurada en la herramienta SonarQube), en la Figura 4-62 se observa la opción para configurar el escáner de SonarQube.

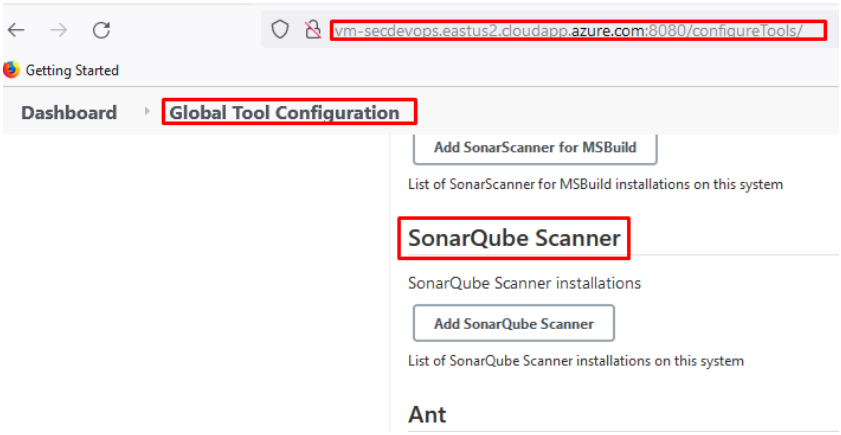



**Figura 4-62.** Configuración SonarQube Scanner – Parte 1

La configuración del escáner de SonarQube se observa en la Figura 4-63.

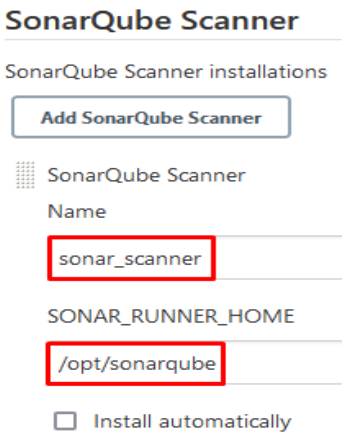



**Figura 4-63.** Configuración SonarQube Scanner – Parte 2

Con el token creado en la herramienta SonarQube se crea la credencial en Jenkins y esta se almacena con el nombre *"SonarQube-Jenkins"* como se observa en la Figura 4-64.

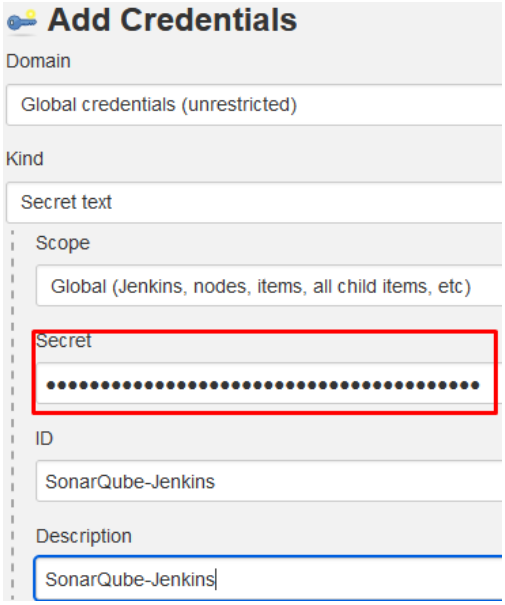



**Figura 4-64.** Configuración credencial SonarQube - Jenkins

Adicionalmente, es necesario configurar la URL del servidor donde se está ejecutando SonarQube, en este caso corresponde a la máquina virtual "vm-secdevops-prod-eastus2-001". Para esto es necesario ir a la *pestaña "manage Jenkins –> Configure Systems –> SonarQube Servers*" configurar el nombre, URL del servidor, y la credencial *"SonarQube-Jenkins"* creada previamente. Tener en cuenta que el nombre del servidor configurado será el utilizado en la tarea de SonarQube del pipeline. En la Figura 4-65 se observa la configuración.

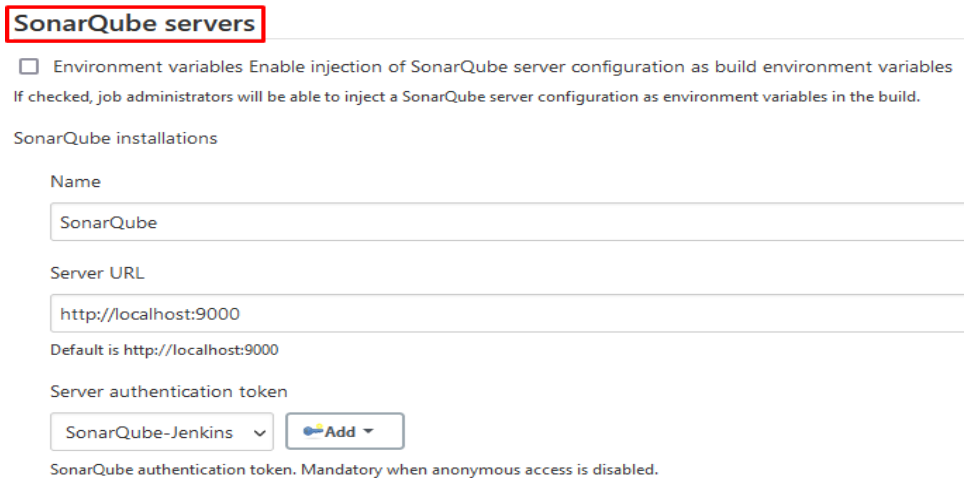



**Figura 4-65**. Configuración servidor SonarQube

*Integración Jenkins-GitLab*

Del mismo modo qué en SonarQube, desde la pestaña *"user settings -->access token"* de GitLab se procede a crear un token de acceso personal (Personal Access Token) con los permisos necesarios para usarlo. Las propiedades del token generado se muestran en la Figura 4-66.

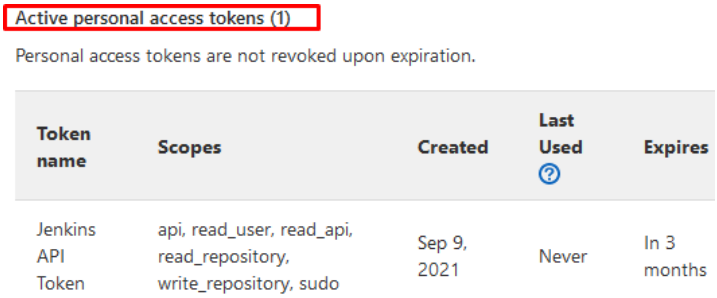



**Figura 4-66.** Propiedades token generado

Ahora se procede a explicar el paso a paso llevado desde Jenkins. Instalación del plugin GitLab y Git como se observa en la Figura 4-67.

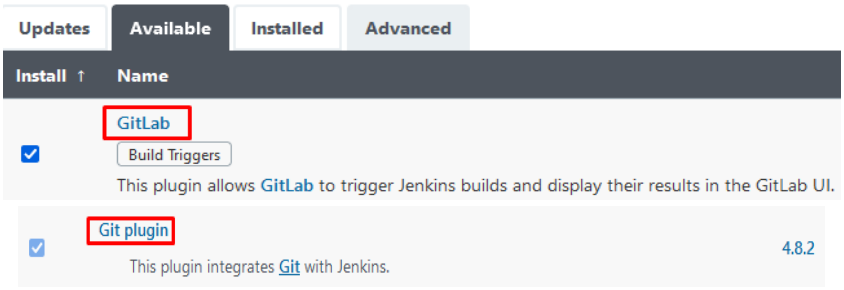



**Figura 4-67**. Plugins instalados en GitLab

Al igual que para SonarQube, es necesario agregar una credencial tipo "GitLab API Token", como se observa en la Figura 4-68.

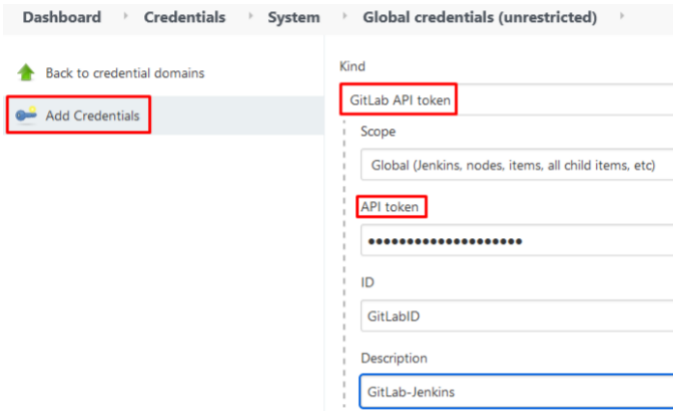



**Figura 4-68**. Credencial creada para GitLab

Para la configuración de la conexión a GitLab se accede a la pestaña "*manage Jenkins → configure systems→ GitLab*". Se asigna la URL donde se ejecuta GitLab "http://192.16.2.4" y se asocia la credencial "*GitLab-Jenkins*" anteriormente creada. Además, se realiza una prueba de conexión, con resultado exitoso. La configuración de la credencial se observa en la Figura 4-69.

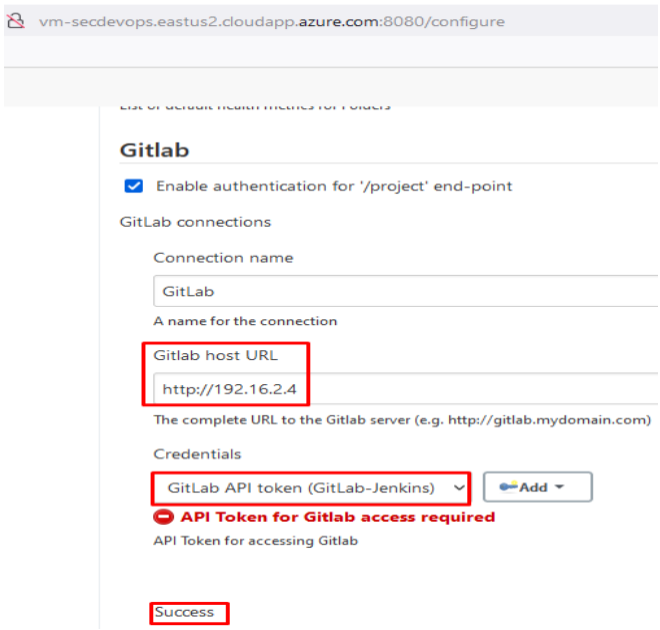



**Figura 4-69.** Creación a GitLab configurada

Adicionalmente, es necesario añadir una nueva credencial para el usuario "root" y contraseña de GitLab, ver Figura 4-70.

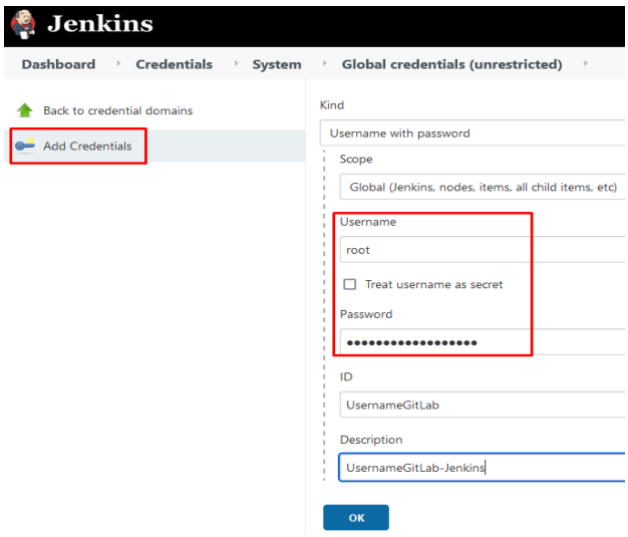



**Figura 4-70**. Creación credencial para el usuario root de GitLab

# D. Anexo: Instalación de las herramientas para la plataforma SecDevOps propuesta

A partir del despliegue de la plataforma DevOps, se procede a exponer el paso a paso para el despliegue de las herramientas propuestas para construir la plataforma SecDevOps.

En este anexo se describe el paso a paso seguido para la instalación de las herramientas de seguridad que se integran a la plataforma DevOps desplegada y configurada en el *Anexo: Instalación de las herramientas para la plataforma DevOps propuesta.*

Partiendo de la instalación de la herramienta SonarQube inicialmente para el análisis de calidad en el código se hace uso de la funcionalidad que contiene adicionalmente para el análisis de seguridad en el código.

***Instalación OWASP Dependency Check***

Inicialmente se debe descargar el plugin "OWASP Dependency-Check Plugin" en Jenkins como se observa en la Figura 4-71.

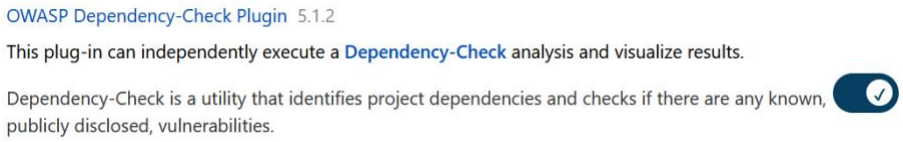



**Figura 4-71.** Plugin instalado en Jenkins.

En la pestaña "Manage Jenkins –> Global Tool Configuration" de Jenkins se debe configurar la instalación automática de la herramienta, como se muestra en la Figura 4-72.

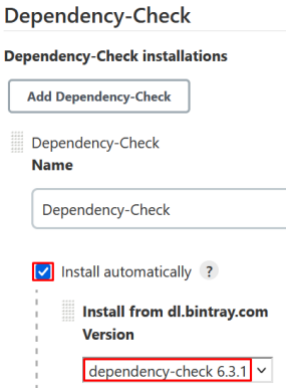



**Figura 4-72.** Configuración instalación automática de la herramienta.

En este punto se completa la configuración de la herramienta para el análisis de dependencias y análisis de composición de software.

- ***Instalación Anchore***

La instalación y configuración se realiza en la máquina virtual "vm-secdevops-prod-eastus2-001". Por ende, es necesario ajustar el tamaño de la máquina a 4vCPU y 16GB de RAM.

Se inicia con la instalación de la herramienta "docker-compose", herramienta empleada para la definición y ejecución de aplicaciones multi-contenerizadas. Luego de seguir el paso a paso definido en [151] se comprueba la instalación, como se muestra en la Figura 4-73.

```
root@vm-gitlab-prod-eastus2-001:/home/admin_devsecops# docker-compose version
docker-compose version 1.25.0, build unknown
docker-py version: 4.1.0
CPython version: 3.8.10
OpenSSL version: OpenSSL 1.1.1f  31 Mar 2020
```

**Figura 4-73.** Validación instalación docker-compose

La herramienta Anchore se conforma de (2) elementos: Anchore-engine y anchore-cli. Se procede a la instalación de Anchore-engine mediante docker-compose. Primero, se crea un directorio donde se va a descargar el manifiesto ".yaml" con la configuración básica de la herramienta, y luego se instala la herramienta mediante la ejecución dentro del directorio creado con el comando "docker-compose up -d", así como se muestra en la Figura 4-73 y Figura 4-74.

```
root@vm-gitlab-prod-eastus2-001:/home/admin_devsecops/anchore# curl -O https://engine.anchore.io/
docs/quickstart/docker-compose.yaml
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  4331    0  4331    0     0  52180      0 --:--:-- --:--:-- --:--:-- 52180
root@vm-gitlab-prod-eastus2-001:/home/admin_devsecops/anchore# ls -la
total 16
drwxr-xr-x 2 root           root           4096 Jun  9 04:37 .
drwxr-xr-x 6 admin_devsecops admin_devsecops 4096 Jun  9 04:35 ..
-rw-r--r-- 1 root           root           4331 Jun  9 04:37 docker-compose.yaml
```

**Figura 4-74**. Descarga manifiesto de Anchore en el directorio

```
root@vm-gitlab-prod-eastus2-001:/home/admin_devsecops/anchore# docker-compose up -d
Creating network "anchore_default" with the default driver
Creating anchore_db_1 ... done
Creating anchore_catalog_1 ... done
Creating anchore_api_1           ... done
Creating anchore_queue_1         ... done
Creating anchore_policy-engine_1 ... done
Creating anchore_analyzer_1      ... done
```

**Figura 4-75**. Instalación de Anchore

Luego de finalizar la instalación de Anchore se procede a validar el estado de los contenedores creados y de los servicios desplegados. Como se expone en las Figura 4-76 y Figura 4-77.

```
root@vm-gitlab-prod-eastus2-001:/home/admin_devsecops/anchore# docker-compose ps
        Name                      Command               State                        Ports
-------------------------------------------------------------------------------------------------------------
anchore_analyzer_1        /docker-entrypoint.sh anch ...   Up (healthy)   8228/tcp
anchore_api_1             /docker-entrypoint.sh anch ...   Up (healthy)   0.0.0.0:8228->8228/tcp,:::8228->8228/tcp
anchore_catalog_1         /docker-entrypoint.sh anch ...   Up (healthy)   8228/tcp
anchore_db_1              docker-entrypoint.sh postgres    Up (healthy)   5432/tcp
anchore_policy-engine_1   /docker-entrypoint.sh anch ...   Up (healthy)   8228/tcp
anchore_queue_1           /docker-entrypoint.sh anch ...   Up (healthy)   8228/tcp
```

**Figura 4-76.** Validación contenedores

```
root@vm-gitlab-prod-eastus2-001:/home/admin_devsecops/anchore# docker-compose exec api anchore-cli system status
Service analyzer (anchore-quickstart, http://analyzer:8228): up
Service simplequeue (anchore-quickstart, http://queue:8228): up
Service policy_engine (anchore-quickstart, http://policy-engine:8228): up
Service apiext (anchore-quickstart, http://api:8228): up
Service catalog (anchore-quickstart, http://catalog:8228): up

Engine DB Version: 0.0.15
Engine Code Version: 1.0.0
```

**Figura 4-77.** Instalación estado Anchore-engine

Se continua con la instalación de anchore-cli siguiendo la guía expuesta en [152].Se ejecutan los siguientes comandos recomendados y la validación de la instalación se puede observar en la Figura 4-78.

*apt-get update*
*apt-get install python3-pip*
*pip install anchorecli*
*Note make sure ~/.local/bin is part of your PATH or just export it directly: export PATH="$HOME/.local/bin/:$PATH"*

```
Successfully installed Click-8.0.1 PyYAML-5.4.1 anchorecli-0.9.3 charset-normalizer-2.0.12 prettytable-2.2.0 python-d
ateutil-2.8.2 requests-2.26.0 six-1.16.0 urllib3-1.26.6 wcwidth-0.2.5
root@vm-gitlab-prod-eastus2-001:/home/admin_devsecops/anchore# export PATH="$HOME/.local/bin/:$PATH"
root@vm-gitlab-prod-eastus2-001:/home/admin_devsecops/anchore# anchore-cli
Usage: anchore-cli [OPTIONS] COMMAND [ARGS]...

Options:
  --config TEXT       Set the location of the anchore-cli yaml configuration
                      file
  --debug             Debug output to stderr
  --u TEXT            Username (or use environment variable ANCHORE_CLI_USER)
  --p TEXT            Password (or use environment variable ANCHORE_CLI_PASS)
  --url TEXT          Service URL (or use environment variable
                      ANCHORE_CLI_URL)
  --hub-url TEXT      Anchore Hub URL (or use environment variable
                      ANCHORE_CLI_HUB_URL)
  --api-version TEXT  Explicitly specify the API version to skip checking.
                      Useful when swagger endpoint is inaccessible
  --insecure          Skip SSL cert checks (or use environment variable
                      ANCHORE_CLI_SSL_VERIFY=<y/n>)
  --json              Output raw API JSON
  --as-account TEXT   Set account context for the command to another account
                      than the one the user belongs to. Subject to authz
  --version           Show the version and exit.
  -h, --help          Show this message and exit.
```

**Figura 4-78.** Validación instalación anchore-cli

Para comprobar el funcionamiento de la herramienta Anchore se procede a validar el estado mediante anchore-cli y a realizar el cambio de contraseña por defecto que se crea en la instalación. El resultado de cada de estas validaciones se pueden ver en las Figura 4-79, Figura 4-80 y Figura 4-81 respectivamente.

```
root@vm-gitlab-prod-eastus2-001:/home/admin_devsecops/anchore# anchore-cli system status
"Unauthorized"
root@vm-gitlab-prod-eastus2-001:/home/admin_devsecops/anchore#
```

**Figura 4-79.** Validación de Anchore mediante anchore-cli

```
root@vm-gitlab-prod-eastus2-001:/home/admin_devsecops/anchore# anchore-cli --u admin --p foobar account user setpassword p
Password (re)set success
```

**Figura 4-80.** Cambio contraseña para el usuario “admin”

```
root@vm-gitlab-prod-eastus2-001:/home/admin_devsecops/anchore# anchore-cli --u admin --p p       system status
Service catalog (anchore-quickstart, http://catalog:8228): up
Service analyzer (anchore-quickstart, http://analyzer:8228): up
Service simplequeue (anchore-quickstart, http://queue:8228): up
Service policy_engine (anchore-quickstart, http://policy-engine:8228): up
Service apiext (anchore-quickstart, http://api:8228): up

Engine DB Version: 0.0.15
Engine Code Version: 1.0.0
root@vm-gitlab-prod-eastus2-001:/home/admin_devsecops/anchore# anchore-cli --u admin --p foobar system status
"Unauthorized"
root@vm-gitlab-prod-eastus2-001:/home/admin_devsecops/anchore#
```

**Figura 4-81.** Validación estado de los servicios con el usuario “admin”

Es importante agregar la cadena de conexión del ACR al Anchore para así el identifique donde debe almacenar la imagen luego de su creación y su análisis de seguridad, Figura 4-82.

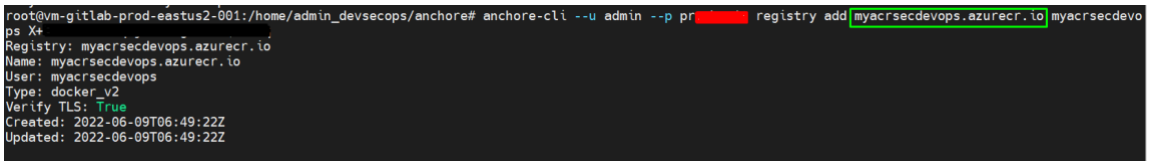



**Figura 4-82**. Adición ACR a Anchore

Por el lado del componente Jenkins es necesario instalar el plugin "Anchore Container Image Scanner", como se muestra en la Figura 4-83.

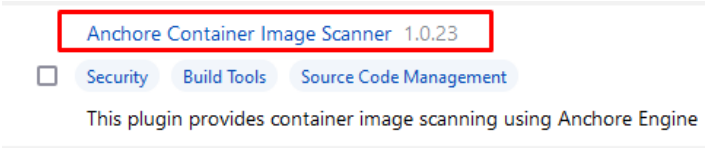



**Figura 4-83**. Instalación plugin de Anchore en Jenkins

Y la configuración de los parámetros para el consumo del Anchore, como se expone en la Figura 4-84.

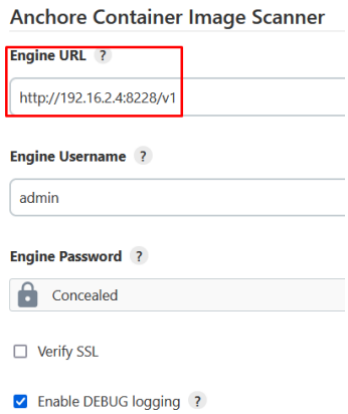



**Figura 4-84**. Configuración parámetros para el consumo de Anchore

### *Instalación OWASP ZAP*

La ejecución de esta herramienta basada en contenerización se realiza en la máquina "vm-secdevops-prod-eastus2-001" mediante Docker. La principal configuración se debe realizar en Jenkins como se expone a continuación. Dado a que la ejecución de la herramienta se realiza mediante un script en una máquina diferente a la máquina del componente Jenkins, es necesario instalar el plugin como se muestra en Figura 4-85.

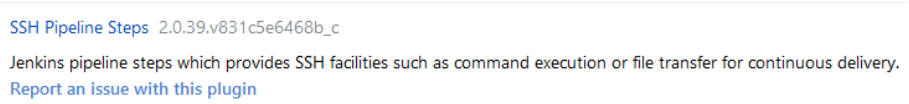



**Figura 4-85**. Instalación plugin SSH en Jenkins

Debido a que se ejecuta en otra máquina, la configuración importante se debe realizar en el paso o step "DAST" del pipeline. A continuación, se muestran algunos parámetros que se deben tener en cuenta, ver Figura 4-86.

```
script  {
  def data = "sudo docker stop owasp2\nsudo docker rm owasp2'
  def remote = [:]
  remote.name = 'Zap-Server'
  remote.host = '192.16.2.4'
  remote.user = 'admin_devsecops'
  remote.password = '@@Admin_DevSecOps**'
  remote.allowAnyHosts = true
  writeFile file: 'baseline-scan-zap.sh', text: data
  sshScript remote: remote, script: "baseline-scan-zap.sh"
```

**Figura 4-86.** Configuración pipeline para ejecución DAST

- ***Instalación Kubescape***

Esta herramienta se instala mediante el siguiente código en la máquina donde se ejecuta el componente Jenkins, como se observa en la Figura 4-87.

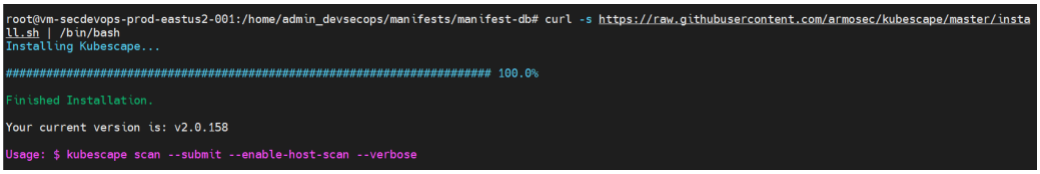



**Figura 4-87.** Instalación Kubescape

# E. Anexo: Instalación de las herramientas usadas en la prueba dos (2)

Dentro de las pruebas propuestas para la validación y verificación de la estrategia de seguridad desarrollada en la tesis, se propone una prueba para validar en ciertos puntos del flujo DevOps, entre ellos, SAST y DAST, con herramientas convencionales y usadas en el mercado. Partiendo de esto se expone el procedimiento llevado para el despliegue y configuración de las herram8ientas usadas para llevar acabo la prueba propuesta.

Las herramientas AppScan on Cloud, Nessus y Qualys son herramientas de licencia paga, por lo que fue necesario crear una cuenta para adquirir la prueba gratis durante un tiempo limitado y así ejecutar el análisis correspondiente. Para la herramienta Qualys se hizo necesario el apoyo de un compañero para la ejecución de la herramienta con la licencia determinada, finalizando, la herramienta Nikto es una herramienta gratuita que está dentro de la suite de Kali Linux. Partiendo de esto, se expone a continuación la instalación, configuración y ejecución de cada una de las herramientas mencionadas.

- ***Instalación AppScan on Cloud***

Primero se crea una cuenta en: https://hclcust.okta.com/ y luego se inicia sesión para ingresar a las opciones desde la pantalla principal. Como se muestra en la Figura 4-88 y Figura 4-89.

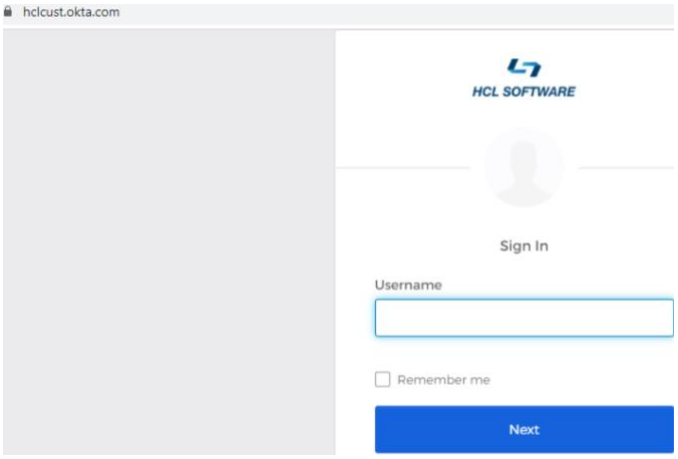



**Figura 4-88.** Inicio sesión AppScan on Cloud

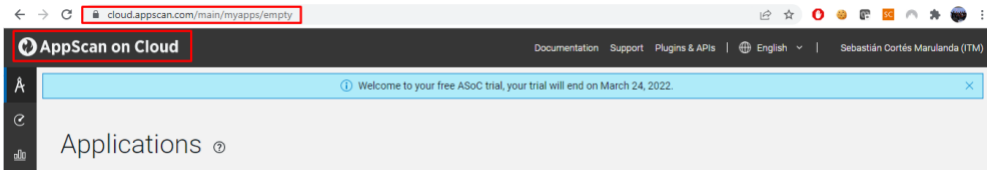



**Figura 4-89.** Pantalla principal AppScan on Cloud

Luego es necesario crear una llave API para configurar la conexión desde Jenkins, ver Figura 4-90.

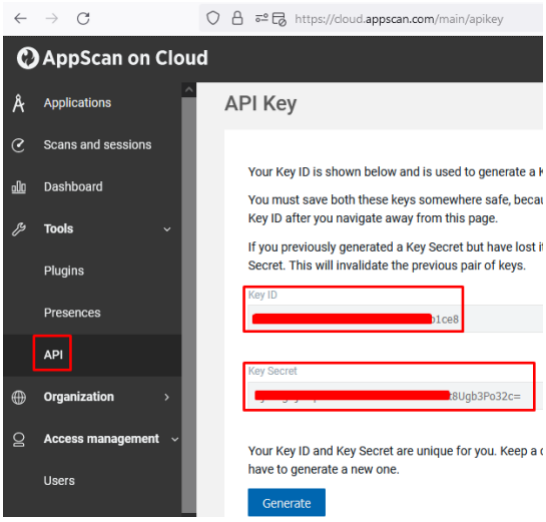



**Figura 4-90.** Creación API desde AppScan On Cloud

En Jenkins se procede a instalar el plugin "HCL_AppScan" y a crear la credencial con la llave API generada previamente. El resultado de esto se expone en la Figura 4-91 y Figura 4-92.

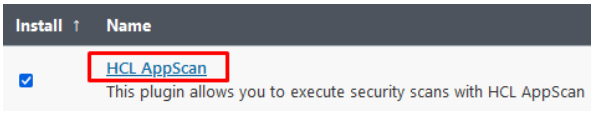



**Figura 4-91**. Instalación plugin AppScan on Cloud

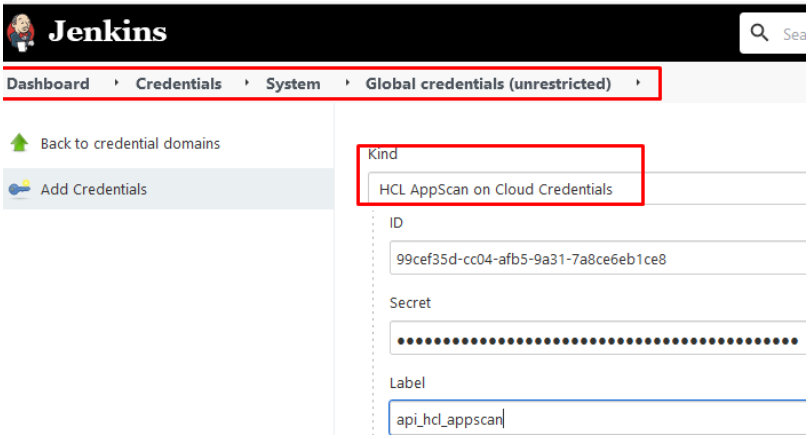



**Figura 4-92.** Credencial para AppScan on Cloud

- ***Instalación Nessus***

La versión usada es nessus essentials. Para descargar el ejecutable es necesario obtener un código de activación que se envía al correo corporativo registrado. Esto se puede ver en la Figura 4-93.

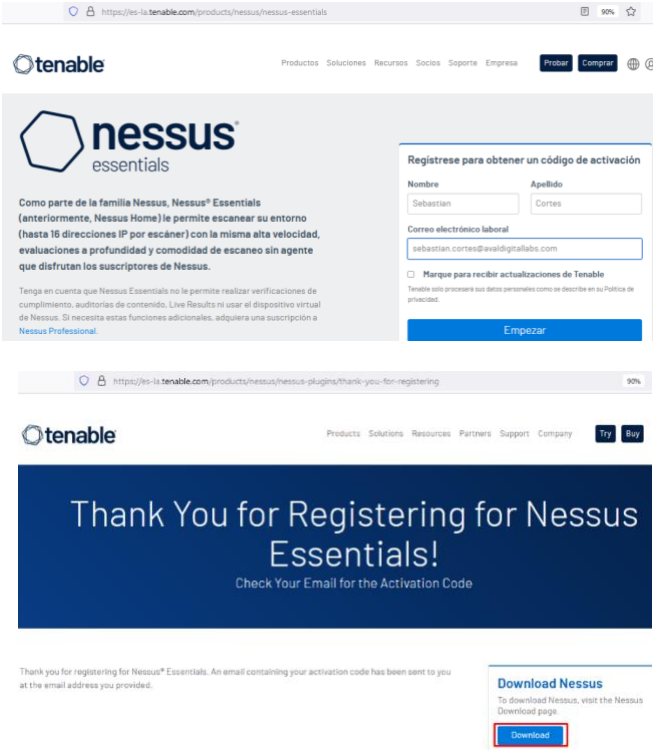



**Figura 4-93.** Registro para obtener el código de activación

El código de validación es enviado al correo registrado, ver Figura 4-94.

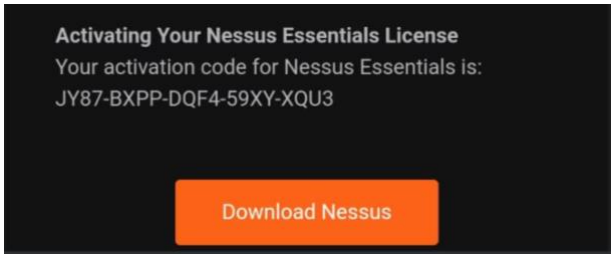



**Figura 4-94.** Código de activación

Luego se procede a la descarga del ejecutable y a la instalación de este. Como se muestra en la Figura 4-95.

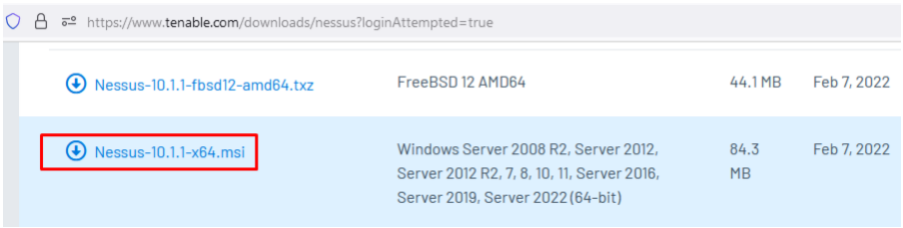


**Figura 4-95.** Descarga e inicio de instalación

Al finalizar la instalación se procede a la configuración inicial de Nessus, donde se requiere seleccionar la versión, el código de activación y la creación del usuario y contraseña. Ver Figura 4-96.

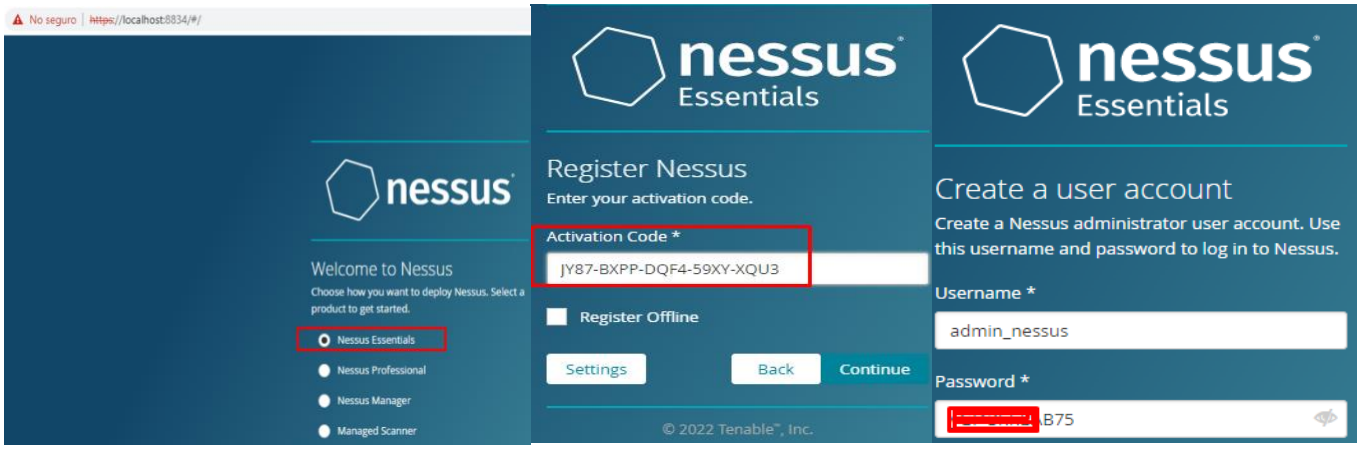



**Figura 4-96.** Instalación Nessus

Cuando la instalación es exitosa se accede a la consola inicial donde se pueden visualizar los diferentes tipos de escaneo que ofrece la herramienta, como se muestra en la Figura 4-97.

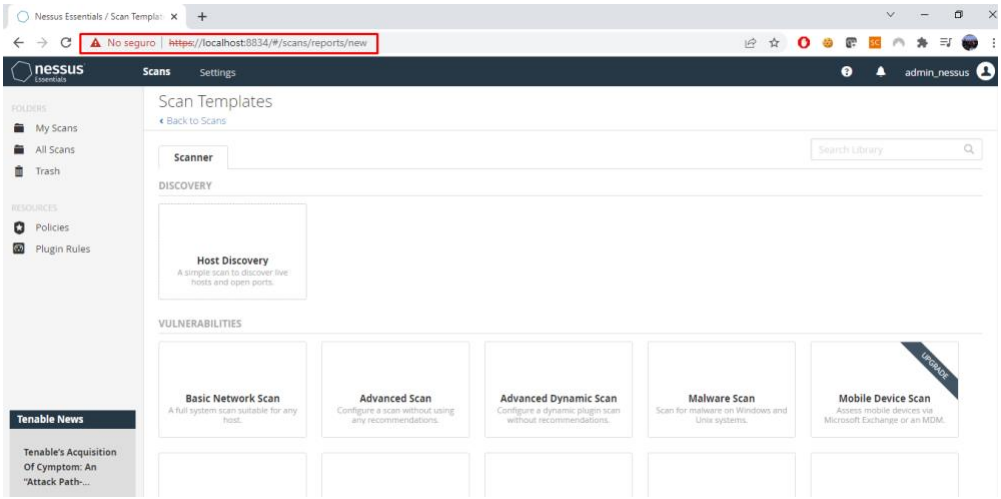



**Figura 4-97.** Pantalla principal Nessus

- ***Instalación Nikto***

Partiendo de la configuración y ejecución de Kali Linux en virtual box se tiene acceso a la herramienta Nikto. La versión del Kali Linux instalado y la visualización de la herramienta se pueden observar en la Figura 4-98 y Figura 4-99.

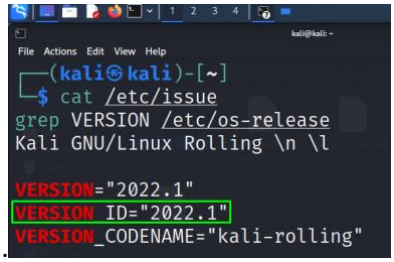



**Figura 4-98.** Versión Kali Linux usada

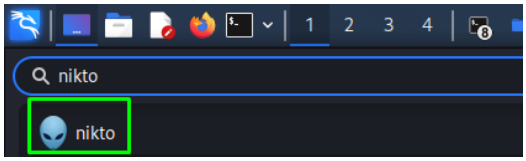



**Figura 4-99.** Nikto en Kali Linux

# Bibliografía

[1] GCFAprendelibre, «¿Qué son las aplicaciones?,» 2020. [En línea]. Available: https://edu.gcfglobal.org/es/cultura-tecnologica/que-son-las-aplicaciones-o-programas/1/. [Último acceso: 20 Agosto 2020].

[2] marianogendra, «Aplicaciones Web Vs Aplicaciones de Escritorio,» 2020. [En línea]. Available: https://www.marianogendra.com.ar/Articulos/aplicaciones-web-vs-escritorio. [Último acceso: 20 Agosto 2020].

[3] Softcorp, «Definición y cómo funcionan las aplicaciones móviles,» 2010. [En línea]. Available: https://servisoftcorp.com/definicion-y-como-funcionan-las-aplicaciones-moviles/#:~:text=Si%20hablamos%20de%20la%20definici%C3%B3n,informado%2C%20entre%20otro%20universo%20de. [Último acceso: 20 Agosto 2020].

[4] J. Rojas, «Vulnerabilidades de aplicaciones web según owasp,» *Universidad Piloto de Colombia,* pp. 1-11, 2018.

[5] Instituto Tecnologico de Matehuala, «2.1 Arquitectura de las aplicaciones Web,» 2015. [En línea]. Available: https://programacionwebisc.wordpress.com/2-1-arquitectura-de-las-aplicaciones-web/. [Último acceso: 12 Agosto 2020].

[6] E. G. Maida y J. Pacienzia, «"Metodologías de Desarrollo de Software," Tesis Final de Licenciatura en Sistemas y Computación,» *Fac. Quim. e Ing "Fray Rogelio Bacon" Pontificia Univ Católica Argentina Santa Maria de los Buenos Aires. Buenos Aires,* 2015.

[7] Microsoft Azure, «¿Qué es DevOps?,» 2020. [En línea]. Available: https://azure.microsoft.com/es-es/overview/what-is-devops/. [Último acceso: 25 Agosto 2020].

[8] M. A. B. L. Z. Mojtaba Shanin, «Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices,» *IEEE Access,* vol. 5, pp. 3909 - 3943, 2017.

[9] S. Pittet, «Atlassian CI/CD,» 2020. [En línea]. Available: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment. [Último acceso: 27 Agosto 2020].

[10] AWS, «¿Qué es la entrega continua?,» 2020. [En línea]. Available: https://aws.amazon.com/es/devops/continuous-delivery/?nc1=h_ls. [Último acceso: 27 Agosto 2020].

[11] E. Sánchez, «Introducción a DevOps. Qué es y cómo implementarlo,» Tribalyte Techologies, 10 Mayo 2019. [En línea]. Available: https://tech.tribalyte.eu/blog-introduccion-devops. [Último acceso: 25 Agosto 2020].

[12] K. Morales, «Ciclo de vida del DevOps,» Platzi, 2019. [En línea]. Available: https://platzi.com/blog/ciclo-de-vida-del-devops/. [Último acceso: 25 Agosto 2020].

[13] T. Gourdel, «Building a CI/CD pipeline with Talend and Azure DevOps,» 17 Abril 2019. [En línea]. Available: https://www.talend.com/blog/2019/04/17/building-a-ci-cd-pipeline-with-talend-and-azure-devops/. [Último acceso: 11 Septiembre 2020].

[14] S. Vergara, «¿Implementas algún pipeline CI/CD en tu organización?,» ITDO, 2020. [En línea]. Available: https://www.itdo.com/blog/implementas-algun-pipeline-ci-cd-en-tu-organizacion/#:~:text=%C2%BFQue%20es%20un%20pipeline%20CI,del%20desarrollo%20de%20las%20aplicaciones.. [Último acceso: 10 Septiembre 2020].

[15] L. Bass, R. Holz, P. Rimba, A. B. Tran y L. Zhu, «Securing a Deployment Pipeline,» *IEEE/ACM 3rd International Workshop on Release Engineering,* pp. 4-7, 2015.

[16] J. J. Caño Quintero, «"DevSecOps: integración de herramientas SAST, DAST y de análisis de Dockers en un sistema de integración continua", Máster en Seguridad de las TIC,» *Universitat Oberta de Catalunya, Barcelona, España,* 2019.

[17] G. A. Gómez Zafra, «"Herramientas de prueba de seguridad de aplicaciones", Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones (MISTIC),,» *Universidad Oberta de Catalunya, Barcelona, España,* 2017.

[18] C. Paule, «"Securing DevOps - Detection of vulnerabilities in CDe pipelines", Tésis de Maestría, Institute of Software Technology Reliable Software Systems,» *University of Stuttgart, Alemania,* 2018.

[19] S. I. Ricote, «"Seguridad en el ciclo de vida del desarrollo del software. DevSecOps", Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones, Seguridad Empresarial,» *Universitat Oberta de Catalunya, Barcelona, España,* 2019.

[20] Atlassian, «Bamboo,» 2020. [En línea]. Available: https://www.atlassian.com/es/software/bamboo. [Último acceso: 24 Septiembre 2020].

[21] Buildbot, «Buildbot The Continuous Integration Framework,» 2020. [En línea]. Available: https://buildbot.net/. [Último acceso: 24 Septiembre 2020].

[22] C. Paule, T. F. Dullmann y A. Van Hoorn, «Vulnerabilities in Continuous Delivery Pipelines? A Case Study,» *IEEE International Conference on Software Architecture Companion,* pp. 102-108, 2019.

[23] CIC Consulting Informático, «Seguridad de la Información y Ciberseguridad ¿es lo mismo?,» 14 Enero 2019. [En línea]. Available: https://www.cic.es/seguridad-de-la-informacion-y-ciberseguridad-es-lo-mismo/. [Último acceso: 15 Julio 2020].

[24] Incibe Instituto Nacional de Ciberseguridad, «Amenaza vs Vulnerabilidad, ¿sabes en qué se diferencian?,» 20 Marzo 2017. [En línea]. Available: https://www.incibe.es/protege-tu-

empresa/blog/amenaza-vs-vulnerabilidad-sabes-se-diferencian. [Último acceso: 15 Julio 2020].

[25] ISO27001, «Glosario,» 2005. [En línea]. Available: https://www.iso27000.es/glosario.html. [Último acceso: 10 Septiembre 2020].

[26] Unir Revista, «Claves de las políticas de seguridad informática,» 14 Mayo 2020. [En línea]. Available: https://www.unir.net/ingenieria/revista/noticias/politicas-seguridad-informatica/549204996232/. [Último acceso: 14 Septiembre 2020].

[27] NNT Security Through System Integrity, «Cis Benchmark Hardening/Vulnerability Checklists,» 2020. [En línea]. Available: https://www.newnettechnologies.com/cis-benchmark.html?keyword=Cis%20Benchmark&gclid=CjwKCAjw5Kv7BRBSEiwAXGDEld3h3F1jws4zE_JlqMf7NAiZCVgGL2Zh-f2EMd0j0d9Vn3HvM4diWhoCxlAQAvD_BwE. [Último acceso: 23 Septiembre 2020].

[28] Owasp, «OWASP Top 10 - 2017 Los diez riesgos más críticos en Aplicaciones Web,» 23 Enero 2020. [En línea]. Available: https://wiki.owasp.org/images/5/5e/OWASP-Top-10-2017-es.pdf. [Último acceso: 13 Agosto 2020].

[29] Owasp, «OWASP Top Ten,» 2020. [En línea]. Available: https://owasp.org/www-project-top-ten/#. [Último acceso: 13 Agosto 2020].

[30] OWASP, «Owasp TOP 10,» OWASP Top 10 team, 2021. [En línea]. Available: https://owasp.org/Top10/. [Último acceso: 16 Julio 2022].

[31] OWASP, «Estándar de Verificación de Seguridad en Aplicaciones 3.0.1,» Abril 2017. [En línea]. Available: https://owasp.org/www-pdf-archive/Est%C3%A1ndar_de_Verificaci%C3%B3n_de_Seguridad_en_Aplicaciones_3.0.1.pdf. [Último acceso: 14 Septiembre 2020].

[32] OWASP, «OWASP security Knowledge Framework,» 2020. [En línea]. Available: https://owasp.org/www-project-security-knowledge-framework/. [Último acceso: 15 Septiembre 2020].

[33] GitHub, «OWASP Security Knowledge Framework,» 2020. [En línea]. Available: https://github.com/blabla1337/skf-flask. [Último acceso: 15 Septiembre 2020].

[34] OWASP, «OWASP Web Security Testing Guide,» 2020. [En línea]. Available: https://owasp.org/www-project-web-security-testing-guide/. [Último acceso: 15 Septiembre 2020].

[35] OWASP, «OWASP Web Security Testing Guide,» 2020. [En línea]. Available: https://github.com/OWASP/wstg. [Último acceso: 15 Septiembre 2020].

[36] OWASP, «Cross Site Scripting (XSS),» 2020. [En línea]. Available: https://owasp.org/www-community/attacks/xss/. [Último acceso: 10 Septiembre 2020].

[37] OWASP, «Cross Site Request Forgery (CSRF),» 2020. [En línea]. Available: https://owasp.org/www-community/attacks/csrf. [Último acceso: 10 Septiembre 2020].

[38] D. L. Regos, «Seguridad en aplicaciones Web, Máster interuniversitario en seguridad de las tecnologías. Aréa: Seguridad en servicios y aplicacione. Universitat oberta de catalunya,» Barcelona, España, 2015.

[39] OWASP, «SQL Injection,» 2020. [En línea]. Available: https://owasp.org/www-community/attacks/SQL_Injection. [Último acceso: 10 Septiembre 2020].

[40] Netsparker Logo - Web Application Security Scanner, «Remote Code Evaluation (Execution) Vulnerability,» 2020. [En línea]. Available: https://www.netsparker.com/blog/web-security/remote-code-evaluation-execution/. [Último acceso: 15 Septiembre 2020].

[41] OWASP, «Broken Access Control,» 2020, [En línea]. Available: https://owasp.org/www-community/Broken_Access_Control. [Último acceso: 15 Septiembre 2020].

[42] welivesecurity, «¿Qué es un 0-day?,» 25 Febrero 2015. [En línea]. Available: https://www.welivesecurity.com/la-es/2015/02/25/que-es-un-0-day/. [Último acceso: 22 Octubre 2020].

[43] GitHub, «DefectDojo,» 2020. [En línea]. Available: https://github.com/DefectDojo/django-DefectDojo. [Último acceso: 15 Septiembre 2020].

[44] J. M. O. Candel, Desarrollo seguro en ingeniería del software. Aplicaciones seguras con Android, NodeJS, Python y C++, España: Marcombo, 2020.

[45] SonarQube, «SonarQube,» 2020, [En línea]. Available: https://docs.sonarqube.org/latest/. [Último acceso: 15 Septiembre 2020].

[46] GitHub, «OWASP Find Security Bug,» 2020. [En línea]. Available: https://github.com/find-sec-bugs/find-sec-bugs. [Último acceso: 15 Septiembre 2020].

[47] LGTM, «About LGTM,» 2020, [En línea]. Available: https://lgtm.com/help/lgtm/about-lgtm. [Último acceso: 15 Septiembre 2020].

[48] Sonatype, «OSS INDEX,» 2018. [En línea]. Available: https://ossindex.sonatype.org/. [Último acceso: 15 Septiembre 2020].

[49] snyk, «Snyk basics,» 2020. [En línea]. Available: https://support.snyk.io/hc/en-us/articles/360011378558-Introduction-to-Snyk. [Último acceso: 15 Septiembre 2020].

[50] CVE, «CVE,» 14 Septiembre 2020. [En línea]. Available: https://cve.mitre.org/. [Último acceso: 15 Septiembre 2020].

[51] CWE, «CWE Common Weakness Enumeration,» 19 Agosto 2019. [En línea]. Available: https://cwe.mitre.org/. [Último acceso: 15 Septiembre 2020].

[52] Inicibe-cert, «Métricas de evaluación de vulnerabilidades: CVSS 3.0,» 21 Julio 2015. [En línea]. Available: https://www.incibe-cert.es/blog/cvss3-0. [Último acceso: 15 Septiembre 2020].

[53] NIST, «National Vulnerability Database (NVD),» 19 Marzo 2018. [En línea]. Available: https://www.nist.gov/programs-projects/national-vulnerability-database-nvd. [Último acceso: 15 Septiembre 2020].

[54] «VulDB,» 2020. [En línea]. Available: https://vuldb.com/es/. [Último acceso: 19 Septiembre 2020].

[55] circl.lu, «cve-search Common Vulnerabilities and Exposures (CVE),» 2020. [En línea]. Available: https://www.circl.lu/services/cve-search/. [Último acceso: 23 Septiembre 2020].

[56] Red Hat, «DevSecOps y la seguridad de DevOps,» 2020. [En línea]. Available: https://www.redhat.com/es/topics/devops/what-is-devsecops.

[57] T-Evolvers, «¿Son iguales las prácticas DevSecOps y SecDevOps? aquí lo resolveremos en detalle,» 27 Diciembre 2019 . [En línea]. Available: https://medium.com/@TEvolvers.dev/son-lo-mismo-las-pr%C3%A1cticas-devsecops-y-secdevops-aqu%C3%AD-te-explicamos-sus-similitudes-y-diferencias-4fbde5d1aef3. [Último acceso: 22 Septiembre 2020].

[58] SANS AppSec, «Secure DevOps Toolchain,» 2018. [En línea]. Available: https://www.sans.org/security-resources/posters/securing-web-application-technologies-swat/60/download. [Último acceso: 22 Septiembre 2018].

[59] D. A., «¿Cómo usar Git Hooks?,» 11 Diciembre 2019. [En línea]. Available: https://www.hostinger.co/tutoriales/como-usar-git-hooks/. [Último acceso: 22 Septiembre 2020].

[60] BeyondTrust, «Secrets Management,» 2020. [En línea]. Available: https://www.beyondtrust.com/resources/glossary/secrets-management#:~:text=Secrets%20management%20refers%20to%20the,parts%20of%20the%20IT%20ecosystem.. [Último acceso: 23 Septiembre 2020].

[61] B. Olivares y G. Eduardo, «"Modelado de amenazas, una técnica de análisis y gestión de riesgo asociado a software y aplicaciones",,» *Universidad Piloto de Colombia.*

[62] OWASP, «OWASP Proactive Controls,» 2020. [En línea]. Available: https://owasp.org/www-project-proactive-controls/. [Último acceso: 22 Septiembre 2020].

[63] Gartner Glossary, «Static Application Security Testing (SAST),» 2020. [En línea]. Available: https://www.gartner.com/en/information-technology/glossary/static-application-security-testing-sast. [Último acceso: 22 Septiembre 2020].

[64] revenera blog, «What is Software Composition Analysis?,» 2020. [En línea]. Available: https://www.revenera.com/blog/what-is-software-composition-analysis/. [Último acceso: 22 Septiembre 2020].

[65] Imperva, «Vulnerability assessment,» 2020. [En línea]. Available: https://www.imperva.com/learn/application-security/vulnerability-assessment/#:~:text=A%20vulnerability%20assessment%20is%20a,mitigation%2C%20if%20and%20whenever%20needed.. [Último acceso: 23 Septiembre 2020].

[66] Tenable, «Gestión de vulnerabilidades: Todo lo que debe saber,» 2020. [En línea]. Available: https://es-la.tenable.com/vulnerability-management. [Último acceso: 23 Septiembre 2020].

[67] Platzi, «Docker Security Scanning: ¿Qué es y para qué sirve?,» 2016. [En línea]. Available: https://platzi.com/blog/docker-security-scanning/. [Último acceso: 23 Septiembre 2020].

[68] welivesecurity, «Penetration Test, ¿en qué consiste?,» 24 Julio 2012. [En línea]. Available: https://www.welivesecurity.com/la-es/2012/07/24/penetration-test-en-que-consiste/. [Último acceso: 23 Septiembre 2020].

[69] BeyondTrust, «Systems Hardening,» 2020. [En línea]. Available: https://www.beyondtrust.com/resources/glossary/systems-hardening. [Último acceso: 23 Septiembre 2020].

[70] Gartner, «Dynamic Application Security Testing (DAST),» 2020. [En línea]. Available: https://www.gartner.com/en/information-technology/glossary/dynamic-application-security-testing-dast. [Último acceso: 23 Septiembre 2020].

[71] Veracode, «Benefits of a DAST test for application security,» 2020. [En línea]. Available: https://www.veracode.com/security/dast-test. [Último acceso: 23 Septiembre 2020].

[72] BlackDuck, «Black Duck by Synopsys,» 2020. [En línea]. Available: https://www.blackducksoftware.com/. [Último acceso: 24 Septiembre 2020].

[73] X. Fernández Ginés , «"Seguridad en Docker", Máster en Seguridad de las TIC,,» *Máster en Seguridad de las TIC, Barcelona, España,* 2018.

[74] J. Aladrén García, «"Análisis estático de vulnerabilidades en Kubernetes para entornos de integración continua", Grado en Telecomunicaciones Telemática,» *Universidad Oberta de Catalunya, Barcelona, España,* 2020.

[75] phoenixNAP, «17 Best Vulnerability Assessment Scanning Tools,» 2020. [En línea]. Available: https://phoenixnap.com/blog/vulnerability-assessment-scanning-tools. [Último acceso: 15 Octubre 2020].

[76] S. M. T. N. J. C. Kelly Brady, «Docker Container Security in Cloud Computing,» *IEEE, Dep. Computer an Cyber Sciences,* pp. 975-980, 2020.

[77] M. Koopman, «"A Framework for Detecting and Preventing Security Vulnerabilities in Continuous Integration/Continuous Delivery pipelines", Department of Services, Cybersecurity & Safety,» *University of Twente, Enschede, Países Bajos,* 2019.

[78] N. Beigi-Mohammadi, M. Litoiu, M. Emami-Taba, L. Tahvildari, M. Fokaefs, E. Merlo y I. V. Onut, «A DevOps Framework for Quality-Driven Self-Protection in Web Software Systems,» *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering, Ontario, Canada,* pp. 270-74, 2018.

[79] F. Ullah, M. Shahin, M. Zahedi y M. A. Babar, «Security Support in Continuous Deployment Pipeline,» *International Conference on Evaluation of Novel Approaches to Software Engineering, ENASE, Porto, Portugal,* vol. I, pp. 57-68, 2017.

[80] N. Wilde, B. Eddy, K. Patel, N. Cooper, V. Gamboa, B. Mishra y K. Shah, «Security for DevOps deployment processes: Defenses, risks, research directions,» *International Journal of Software Engineering & Applications (IJSEA), ,* vol. VII, nº 6, pp. 1-16, 2016.

[81] Stack Overflow, «Stack Overflow,» 2020. [En línea]. Available: https://stackoverflow.com/. [Último acceso: 28 Octubre 2020].

[82] J. Widman, «The Most Popular Programming Languages of 2019,» 5 Septiembre 2019. [En línea]. Available: https://blog.newrelic.com/technology/most-popular-programming-languages-of-2019/. [Último acceso: 28 Octubre 2020].

[83] Stack Overflow, «2015 Developer Survey,» 2015. [En línea]. Available: https://insights.stackoverflow.com/survey/2015. [Último acceso: 28 Octubre 2020].

[84] Stack Overflow, «Developer Survey Results 2016,» 2016. [En línea]. Available: https://insights.stackoverflow.com/survey/2016. [Último acceso: 28 Octubre 2020].

[85] Stack Overflow, «Developer Survey Results 2017,» 2017. [En línea]. Available: https://insights.stackoverflow.com/survey/2017. [Último acceso: 28 Octubre 2020].

[86] Stack Overflow, «Developer Survey Results 2018,» 2018. [En línea]. Available: https://insights.stackoverflow.com/survey/2018. [Último acceso: 28 Octubre 2020].

[87] Stack Overflow, «Developer Survey Results 2019,» 2019. [En línea]. Available: https://insights.stackoverflow.com/survey/2019. [Último acceso: 28 Octubre 2020].

[88] Stack Overflow, «2020 Developer Survey,» 2020. [En línea]. Available: https://insights.stackoverflow.com/survey/2020. [Último acceso: 28 Octubre 2020].

[89] TIOBE The Software Quality Company, «TIOBE Index for October 2020,» 2020. [En línea]. Available: https://www.tiobe.com/tiobe-index/. [Último acceso: 28 Octubre 2020].

[90] Synopsys, «Black Duck by Synopsys,» 2022. [En línea]. Available: https://www.blackducksoftware.com/. [Último acceso: 26 Mayo 2022].

[91] Snyk, «What is Snyk?,» 2022. [En línea]. Available: https://snyk.io/what-is-snyk/. [Último acceso: 26 Mayo 2022].

[92] Github, «Talisman,» 2022. [En línea]. Available: https://thoughtworks.github.io/talisman/. [Último acceso: 26 Mayo 2022].

[93] M. Hudson, «Git Hooks,» Github, 2022. [En línea]. Available: https://githooks.com/. [Último acceso: 26 Mayo 2022].

[94] M. Finkle, «pre-commit,» 2022. [En línea]. Available: https://pre-commit.com/. [Último acceso: 26 Mayo 2022].

[95] Github, «detect-secrets,» 2022. [En línea]. Available: https://github.com/Yelp/detect-secrets. [Último acceso: 26 Mayo 2022].

[96] Github, [En línea]. Available: https://github.com/ezekg/git-hound. [Último acceso: 26 Mayo 2022].

[97] Carlos, «Docker Security Scanning: ¿Qué es y para qué sirve?,» Platzi, 2016. [En línea]. Available: https://platzi.com/blog/docker-security-scanning/. [Último acceso: 26 Mayo 2022].

[98] GitLab, «The One DevOps Platform,» GitLab B.V., 2022. [En línea]. Available: https://about.gitlab.com/. [Último acceso: 26 Mayo 2022].

[99] Atlassian, «Compila, prueba e implementa con confianza,» Atlassian, 2022. [En línea]. Available: https://www.atlassian.com/es/software/bamboo. [Último acceso: 26 Mayo 2022].

[100] «The Continuous Integration Framework,» [En línea]. Available: https://buildbot.net/. [Último acceso: 26 Mayo 2022].

[101] Microsoft, «Azure DevOps,» 2022. [En línea]. Available: https://azure.microsoft.com/es-es/services/devops/. [Último acceso: 26 Mayo 2022].

[102] GitLab, «GitLab CI/CD,» [En línea]. Available: https://docs.gitlab.com/ee/ci/. [Último acceso: 26 Mayo 2022].

[103] WhiteSource is now Mend, «You Code. We Cure,» White Source Ltd, 2022 . [En línea]. Available: https://www.mend.io/. [Último acceso: 26 Mayo 2022].

[104] CircleCI, «The continuous integration platform preferred by over,» [En línea]. Available: https://circleci.com/. [Último acceso: 26 Mayo 2022].

[105] SonarCloud, «Clean Code,» SonarSource S.A, 2022. [En línea]. Available: https://sonarcloud.io/. [Último acceso: 26 Mayo 2022].

[106] Apache Software Foundation, «Apache® Subversion®,» [En línea]. Available: https://subversion.apache.org/. [Último acceso: 26 Mayo 2022].

[107] New Relic, «Monitor, debug, and improve your entire stack,» New Relic, Inc, 2022. [En línea]. Available: https://newrelic.com/. [Último acceso: 26 Mayo 2022].

[108] The Apache ANT Project, «Apache Ant,» The Apache Software Foundation, 2022. [En línea]. Available: https://ant.apache.org/. [Último acceso: 26 Mayo 2022].

[109] Github, «JavaMelody : monitoring of JavaEE applications,» GitHub, Inc. , 2022. [En línea]. Available: https://github.com/javamelody/javamelody/wiki. [Último acceso: 26 Mayo 2022].

[110] Logstash, «Centralize, transform & stash your data,» Elasticsearch B.V., 2022. [En línea]. Available: https://www.elastic.co/logstash/. [Último acceso: 26 Mayo 2022].

[111] DevOps Dozen, «DevOps Dozen 2017 – Winners,» 2021. [En línea]. Available: https://devopsdozen.com/devops-dozen-2017-winners/. [Último acceso: 26 Mayo 2022].

[112] P. Dhabekar, «Top 7 Best CI/CD Tools you should get your hands on in 2020,» 12 Junio 2020. [En línea]. Available: https://medium.com/devops-dudes/top-7-best-ci-cd-tools-you-should-get-your-hands-on-in-2020-832c29db936a. [Último acceso: 16 Agosto 2020].

[113] GitHub, «Threat Modeling,» OWASP, 2022. [En línea]. Available: https://github.com/OWASP/DevSecOpsGuideline/blob/master/document/00b-Threat-modeling.md. [Último acceso: 30 Mayo 2022].

[114] OWASP, «OWASP Threat Dragon,» OWASP Open Web Application Security Project, 2022. [En línea]. Available: https://owasp.org/www-project-threat-dragon/. [Último acceso: 30 Mayo 2022].

[115] GitHub, « www-project-proactive-controls,» OWASP, 2022. [En línea]. Available: https://github.com/OWASP/www-project-proactive-controls/blob/master/v3/OWASP_Top_10_Proactive_Controls_V3.pdf. [Último acceso: 30 Mayo 2022].

[116] OWASP, «OWASP Application Security Verification Standard,» Open Web Application Security Project, 2022. [En línea]. Available: https://owasp.org/www-project-application-security-verification-standard/. [Último acceso: 30 Mayo 2022].

[117] GitHub, «OWASP Application Security Verification Standard,» 2022. [En línea]. Available: https://github.com/OWASP/ASVS/tree/v4.0.3#latest-stable-version---403. [Último acceso: 30 Mayo 2022].

[118] GitHub, «Take care secrets and credentials in repositories,» OWASP, 2022. [En línea]. Available: https://github.com/OWASP/DevSecOpsGuideline/blob/master/document/01a-Secrets-Management.md. [Último acceso: 30 Mayo 2022].

[119] «¿Qué es el análisis de composición de software?,» 6 Noviembre 2021. [En línea]. Available: https://tecnologiandroid.com/que-es-el-analisis-de-composicion-de-software/. [Último acceso: 30 Mayo 2022].

[120] B-Secure, «Ciclo Continuo de Gestión de Vulnerabilidades,» 2022, [En línea]. Available: https://www.b-secure.co/ciclo-continuo-de-gestion-de-vulnerabilidades. [Último acceso: 11 Junio 2022].

[121] Suzanna Haworth, «Matriz RACI Simplificado: Cómo Crear una Matriz de Responsabilidades que Realmente Funcione,» The Digital Project Manager, 2022. [En línea]. Available: https://thedigitalprojectmanager.com/es/grafico-raci-manera-mas-simple/. [Último acceso: 3 Junio 2022].

[122] «¿Qué es DevSecOps y cuál es su función en la CD?,» JetBrains, 2022. [En línea]. Available: https://www.jetbrains.com/es-es/teamcity/ci-cd-guide/what-is-devsecops/. [Último acceso: 4 Junio 2022].

[123] Isaac Sacolick, «Cuatro estrategias de monitoreo integrado para mejorar las operaciones de TI,» TechTarget, 7 Mayo 2019. [En línea]. Available:

https://www.computerweekly.com/es/consejo/Cuatro-estrategias-de-monitoreo-integrado-para-mejorar-las-operaciones-de-TI. [Último acceso: 4 Junio 2022].

[124] D. Haggerty, «Datadog Signs Definitive Agreement to Acquire Hdiv Security,» Datadog, 05 Mayo 2022. [En línea]. Available: https://investors.datadoghq.com/news-releases/news-release-details/datadog-signs-definitive-agreement-acquire-hdiv-security. [Último acceso: 26 Noviembre 2022].

[125] Trend Micro Incorporated, «¿En qué consiste la vulnerabilidad de Apache Log4J (Log4Shell)?,» 2022. [En línea]. Available: https://www.trendmicro.com/es_es/what-is/apache-log4j-vulnerability.html. [Último acceso: 16 Julio 2022].

[126] CCN-CERT, «CCN-CERT AL 09/21 Vulnerabilidad en Apache Log4j 2,» 13 Julio 2022. [En línea]. Available: https://www.ccn-cert.cni.es/seguridad-al-dia/alertas-ccn-cert/11435-ccn-cert-al-09-21-vulnerabilidad-en-apache-log4j-2.html. [Último acceso: 16 Julio 2022].

[127] J. Pastor, «Log4Shell es la vulnerabilidad crítica 'de proporciones catastróficas' que amenaza con destrozar internet,» Xataka, 13 Diciembre 2021. [En línea]. Available: https://www.xataka.com/seguridad/log4shell-vulnerabilidad-critica-proporciones-catastroficas-que-amenaza-destrozar-internet. [Último acceso: 16 Julio 2022].

[128] G. P. (. M. Group), «Metodología tradicional,» 30 Mayo 2019. [En línea]. Available: https://pmtgrupoeafit.wixsite.com/gestion-proyectos/post/metodolog%C3%ADa-tradicional#:~:text=Las%20metodolog%C3%ADas%20tradicionales%20imponen%20una,de%20desarrollo%20del%20producto%20software.. [Último acceso: 13 Agosto 2020].

[129] G. Lemke, The Software Development Life Cycle and Its Application, Eastern Michigan University, Michigan: Digital Commons, 2018.

[130] Red Hat, «¿Qué es la metodología ágil?,» 2020. [En línea]. Available: https://www.redhat.com/es/devops/what-is-agile-methodology. [Último acceso: 20 Agosto 2020].

[13 1] K. S. y. J. Sutherland, de *La Guía de Scrum*, South American, 2017, p. 22.

[13 2] Socializatte, «Fuera del ámbito del software, como utilizar: Agile & Scrum,» 2020. [En línea]. Available: http://www.socializatte.com/fuera-del-ambito-del-software-como-utilizar-agile-scrum/. [Último acceso: 20 Agosto 2020].

[13 3] mxmendix, «Rapid Application Development (RAD),» [En línea]. Available: https://www.mendix.com/rapid-application-development/. [Último acceso: 24 Octubre 2020].

[13 4] A. Miller, «Secure Software Development Lifecycle (SSDLC),» 21 Junio 2020. [En línea]. Available: https://snyk.io/learn/secure-sdlc/. [Último acceso: 27 Octubre 2020].

[13 5] Microsoft, «Microsoft Security Development Lifecyle (SDL),» 2020. [En línea]. Available: https://www.microsoft.com/en-us/securityengineering/sdl. [Último acceso: 26 Agosto 2020].

[13 6] Us-Cert Cisa, «Correctness by Construction,» 5 Diciembre 2006. [En línea]. Available: https://us-cert.cisa.gov/bsi/articles/knowledge/sdlc-process/correctness-by-construction. [Último acceso: 25 Agosto 2020].

[13 7] Us-Cert Cisa, «Introduction to the CLASP Process,» 16 Noviembre 2006. [En línea]. Available: https://us-cert.cisa.gov/bsi/articles/best-practices/requirements-engineering/introduction-to-the-clasp-process. [Último acceso: 25 Agosto 2020].

[13 8] Gutimar, «Team Software Process (TSP),» 9 Mayo 2012. [En línea]. Available: https://gutimarsoluciones.wordpress.com/2012/05/09/team-software-process-tsp-y-team-process-software-tsp/. [Último acceso: 23 Septiembre 2020].

[13 Oracle, «Importancia de la Software Security Assurance,» 2020. [En línea]. Available: https://www.oracle.com/co/corporate/security-practices/assurance/. [Último acceso: 8 Septiembre 2020].

[14 I. Flechais , C. Mascolo y M. A. Sasse, «Integrating Security And Usability Into The Requirements And Design Process,» *International Journal of Electronic Security and Digital Forensic,* vol. I, nº 1, pp. 13-26, 2007.

[14 J. Pooya, E. Golnaz, S. Mohammad Reza Ayatollahzadeh y S. Babak, «RUPSec : Extending Business Modeling and Requirements Disciplines of RUP for Developing Secure System,» de *EUROMICRO Conference on Software Engineering and Advanced Applications*, Tehran, Iran, 2005.

[14 A. S. Sodiya, S. A. Onashoga y O. B. Ajayi, «Towards Building Secure Software Systems,» *Issues in Informing Science and Information Technology,* vol. III, pp. 636-646, 2006.

[14 Jenkins, «Installing Jenkins,» [En línea]. Available: https://www.jenkins.io/doc/book/installing/linux/#debianubuntu. [Último acceso: 7 Julio 2022].

[14 LinuxTechi, «How to Install and Configure Jenkins on Ubuntu 20.04,» 23 Septiembre 2020. [En línea]. Available: https://www.linuxtechi.com/install-configure-jenkins-ubuntu-20-04/. [Último acceso: 7 Julio 2022].

[14 GitLab, «Install self-managed GitLab,» GitLab B.V, 2022. [En línea]. Available: https://about.gitlab.com/install/?version=ce. [Último acceso: 7 Julio 2022].

[14 SonarQube, «Install the Server,» [En línea]. Available: https://docs.sonarqube.org/9.0/setup/install-server/. [Último acceso: 7 Julio 2022].

[14 Linux Shout, «Install SonarQube on Ubuntu 20.04 LTS Server,» 30 Julio 2021. [En línea]. Available: https://www.how2shout.com/linux/install-sonarqube-on-ubuntu-20-04-18-04-server/. [Último acceso: 7 Julio 2022].

[14 T. Rakwach, «Install SonarQube on Ubuntu 20.04 LTS,» 16 Junio 2021. [En línea]. Available: https://www.vultr.com/docs/install-sonarqube-on-ubuntu-20-04-lts. [Último acceso: 7 Julio 2022].

[14 Docker docs, «Install Docker Engine on Ubuntu,» [En línea]. Available: https://docs.docker.com/engine/install/ubuntu/. [Último acceso: 8 Julio 2022].

[15 D. Soni, «Integrating SonarQube and Jenkins,» 3 Julio 2019. [En línea]. Available: https://aspiresoftware.in/blog/intergrating-sonarqube-and-jenkins/. [Último acceso: 8 Julio 2022].

[15 E. H. Tony Tran, «How To Install and Use Docker Compose on Ubuntu 22.04,» 26 Abril 2022. [En línea]. Available: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-compose-on-ubuntu-22-04. [Último acceso: 26 Julio 2022].

[15 Python Software Foundation, «Anchore Service CLI,» 4 Octubre 2021. [En línea]. Available: https://pypi.org/project/anchorecli/. [Último acceso: 11 Octubre 2022].

[15 OWASP, «OWASP Application Security Verification Standard,» Octubre 2020. [En línea]. Available: https://owasp.org/www-project-application-security-verification-standard/. [Último acceso: 03 Abril 2021].

[15 OWASP, «Application Threat Modeling,» The OWASP Foundation, 2021. [En línea]. Available: https://owasp.org/www-community/Application_Threat_Modeling. [Último acceso: 16 Abril 2021].

[15 D. C, «Threat Modeling for Beginners,» DEV Community, 3 Junio 2020. [En línea]. Available:
5] https://dev.to/caffiendkitten/why-you-need-threat-modeling-3n6p. [Último acceso: 16 Abril 2021].

[15 New Relic, «Monitor, debug, and improve your entire stack,» New Relic, Inc, 2022. [En línea].
6] Available: https://newrelic.com/. [Último acceso: 26 Mayo 2022].