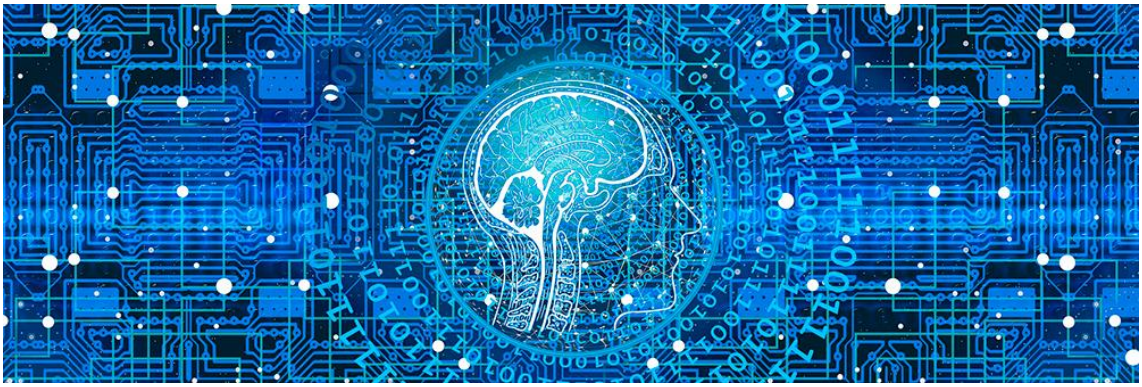
Universitat Oberta de Catalunya

uoc.edu



Desarrollo seguro de *software* bajo la metodología DevSecOps

Borja Varela Gutiérrez
Grado de Ingeniería informática
Seguridad informática

Consultor: *Jorge Miguel Moneo*
Profesora responsable de la asignatura: *Helena Rifá Pous*

Fecha Entrega: 28/12/2021

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Desarrollo seguro de software bajo la metodología DevSecOps*
Nombre del autor:	*Borja Varela Gutiérrez*
Nombre del consultor/a:	*Jorge Miguel Moneo*
Nombre del PRA:	*Helena Rifá Pous*
Fecha de entrega (mm/aaaa):	12/2021
Titulación:	*Grado de Ingeniería Informática*
Área del Trabajo Final:	*Seguridad Informática*
Idioma del trabajo:	*Castellano*
Palabras clave	*Desarrollo, Software, Seguro, DevSecOps*

Resumen del Trabajo:

Rememorando tiempos pasados en que los desarrolladores de aplicaciones centraban sus esfuerzos en diseñar las funcionalidades que debía cumplir una aplicación, asumiendo que el foco las vulnerabilidades se concentraban en la plataforma donde funcionaban...

La finalidad de este trabajo es entender cómo la seguridad de la información ha irrumpido en el diseño de las aplicaciones para generar un nuevo paradigma, dando lugar a nuevas metodologías que se basan en el *security by design*, el cual, permite poner a prueba la seguridad del *software* durante su desarrollo y posterior lanzamiento.

Algunas de estas metodologías son OWASP u OASAM, pero este trabajo se centra en el desarrollo de ***DevSecOps***. Trata de un método ágil que integra de manera natural el entorno seguro y buenas prácticas durante el ciclo de vida de las diferentes aplicaciones. Esta nueva metodología deja atrás las obsoletas prácticas de seguridad que forman parte del pasado.

La metodología ágil se sigue en este proyecto. Se ha dividido en partes más pequeñas, en las que se van trabajando poco a poco, mientras se realiza una labor de investigación. Cada parte se puede modificar o ampliar según avanza el proyecto, transformándose en un proceso circular.

Se ha conseguido plasmar los puntos más importantes para alcanzar la comprensión adecuada de los temas tratados, junto a la puesta en práctica de algunas herramientas.

La seguridad se deja de lado en los desarrollos más de lo que se piensa, por ello, se debe introducir la filosofía de desarrollo seguro como forma de trabajo.

Abstract:

Looking back to old times, when application developers focused their efforts on designing the functionalities that an application had to fulfill, by knowing that the focus of vulnerabilities were in the execution platform.

The purpose of this work is to understand how computer security has had a strong impact into the field of application design to generate a new paradigm, providing new methodologies based on "security by design", which allows testing software security during its development and subsequent launch.

Some of these methods are called OWASP or OASAM, but the work is mainly focused on ***DevSecOps*** development, which is an Agile Method that integrates the safe environment and good practices during the different applications´ life. This new method leaves behind the obsolete security practices.

The agile methodology is the one highlighted in this project, since it has split into smaller parts, which are worked, little by little, while doing research. Each part can be modified or expanded as the project progresses, becoming a circular process.

We have been able to capture the most important points to achieve an adequate understanding of the topics covered, as well as the implementation of some tools.

Security is often neglected in developments more than we think, therefore the philosophy of secure development should be introduced as the main way of working.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

La **seguridad informática** ha ido ganando protagonismo durante los últimos años. Ha pasado de ser un elemento secundario, -donde los desarrolladores no le daban la importancia que merece- a prioritario en la mayoría de los sistemas actuales [1].

En las aplicaciones, se puede afirmar que es uno de los elementos más importantes y que más trabajo puede acarrear durante el desarrollo de las mismas. El gran abanico de elementos y actores que intervienen durante su ejecución, genera un gran desafío para los programadores.

Una violación en la seguridad de una aplicación o sistema, puede poner en riesgo la información crítica de personas o de organizaciones. En el mundo actual, este hecho puede suponer grandes problemas o costes de un alcance insospechado.

Para crear una aplicación segura que cumpla con las expectativas actuales, se debe conocer contra qué o quién se debe defender, por lo que es importante estar informado de los ataques más comunes e ir conociendo los últimos que se van produciendo.

La gran oleada de nuevos dispositivos conectados a Internet ha dado lugar al **IOT** (*Internet Of Things*), el cual nos facilita un sin fin de aspectos de la vida cotidiana. Sin embargo, en paralelo se exponen a posibles ataques para los que estos dispositivos no siempre están preparados.

Por estos y otros motivos, se han desarrollado una serie de **metodologías** que intentan ser una guía útil para los equipos que desarrollan nuevo *software*. Y puedan implantar métodos y buenas prácticas relacionadas con el desarrollo de *software* seguro. Con el fin de alcanzar el objetivo de obtener aplicaciones seguras.

Gracias a esta nueva mentalidad, los diferentes equipos que participan en proyectos de creación de *software* -como pueden ser los programadores y los responsables de la infraestructura- han creado vínculos más fuertes dando lugar a relaciones más estrechas. Esto origina una mayor simbiosis entre los participantes, acabando con disputas de quién es el responsable de que el *software* tenga errores. Ahora, todos reman en la misma dirección.

Se van a presentar diferentes procedimientos que han ido cambiando el diseño, el desarrollo o la programación (entre otros) de las aplicaciones, para centrarse en el nuevo paradigma que se ha generado de manera natural, debido al protagonismo que ha alcanzado la seguridad en el campo de la generación de nuevo *software*. Centrándose en la metodología **DevSecOps** [2].

Del mismo modo, se presentan una serie de recomendaciones y buenas prácticas, que se han ido dando forma con el paso del tiempo, gracias a las experiencias y problemas que han ido viviendo los actores que de una manera u otra, han intervenido en el desarrollo de *software* y aprendiendo sobre la marcha.

1.2 Motivación

La motivación que ha llevado a elegir el tema del TFG sobre la rama de la seguridad informática es porque, en las asignaturas estudiadas a lo largo del grado, donde se ha visto esta temática, han resultado de las más interesantes. Además, la seguridad es un tema muy actual y de continua evolución, por lo que llama mucho la atención.

Mi experiencia en el área de seguridad es prácticamente nula, ya que se basa en las asignaturas cursadas durante el grado, como he mencionado con anterioridad. Pero creo que aunque me cueste y pueda conllevar más esfuerzo por mi parte, será llevadero ya que es una temática que me motiva, y quien sabe, para seguir formándome en la misma y dedicarme laboralmente en este campo en el futuro.

1.3 Estado del arte

En la actualidad, el desarrollo seguro de aplicaciones consiste en aplicar ciertas consideraciones y elementos durante el periodo de tiempo que dura la creación de la aplicación hasta lanzarla a producción. Inclusive, una vez distribuida al público, también se deben seguir un conjunto de buenas prácticas para proteger al *software* de posibles amenazas [5].

La mayoría de las aplicaciones de hoy en día, están conectadas de alguna manera a **Internet**. Desde solicitar una licencia, hasta contar con una infraestructura en *cloud*. Esta situación, aporta grandes cualidades y beneficios al *software,* pero de manera paralela, también lo expone a un sin fin de amenazas capaces de atacar sus vulnerabilidades.

Para luchar contra estas amenazas, los desarrolladores integran diferentes propiedades que forman los pilares de la seguridad para construir aplicaciones robustas y seguras. Por ejemplo:

- **Confidencialidad**: la información solo debe estar a disposición de la persona autorizada para ello.

- **Integridad**: la información no debe ser adulterada (cambiar) salvo que sea de manera expresa con autorización.

- **Disponibilidad**: la información o recursos deben estar disponibles en todo momento para ser consultados o utilizados.

- **Autenticación**: ya sea con el clásico usuario y contraseña o por algún método biométrico, -como la huella digital o el iris-. Con este método se busca conseguir que solo la persona que dice ser quien es acceda al sistema con sus credenciales [6].

  Otras propiedades importantes son:

- **Autenticación multifactor:** se añade otra capa de seguridad a la autenticación solicitando un código o una confirmación, por ejemplo, enviado a un dispositivo móvil, tras realizar la primera autenticación de usuario y contraseña [7].

- **Autorización**: crear una jerarquía de permisos para que usuarios no autorizados no puedan acceder a cierta información.

- **Cifrado**: tanto el cifrado de datos como de las comunicaciones es un elemento muy común para asegurar la integridad de la información [8].

Dependiendo de la arquitectura en la que trabaje la aplicación, en muchas ocasiones puede suponer desafíos adicionales de una gran dificultad a la hora de proporcionar soluciones para que la seguridad no sea violada. Algunos ejemplos son:

- **Aplicaciones web**: son aquellas que se acceden a través de un navegador web, y ofrecen servicios cliente-servidor. Toda esta comunicación se realiza a través de Internet, los datos pueden ser vulnerables mientras viajan por una red pública. Las empresas ponen especial interés en conseguir comunicaciones seguras para sus servicios web.

- **Aplicaciones P2P (*Peer to Peer*)**: se trata de un sistema gratuito con el que se consigue que diferentes *hosts* sean capaces de compartir información a través de Internet. Una de las aplicaciones más populares es BitTorrent, que cuenta con su propio protocolo que se gestiona a través de las redes PRP.

- **Aplicaciones para *smartphones***: de la misma manera que otras aplicaciones, intercambian datos a través de Internet, por lo que se exponen a ataques similares. Las compañías pueden implementar diferentes servicios, -como un MDM (*Mobile Device Management*)- de gestión para desplegar diferentes políticas de seguridad a los terminales y bloquear el acceso con los que no cumplan los requisitos [9].

Se han gestado diferentes **metodologías de seguridad** que se incorporan durante las fases de desarrollo de las aplicaciones. Con el paso del tiempo, la experiencia adquirida mejora y los métodos evolucionan, para proporcionar la seguridad que los diferentes tipos de aplicaciones requieren en la actualidad. Algunas de estas metodologías son OWASP o DevSecOps.

También existen un conjunto de buenas prácticas que se deben llevar a cabo, como, por ejemplo, realizar auditorías de seguridad para conseguir que las aplicaciones

cumplan una serie de expectativas para que se pueda catalogar como segura. Suele consistir en intentos de intrusión, que imitan el *modus operandi* que suelen utilizar los ciberdelincuentes. También se realizan pruebas de ingeniería social o *phishing*, entre otros [7].

1.4 Objetivos del Trabajo

A continuación, se presentan los objetivos que se pretenden alcanzar con el análisis de las principales metodologías para obtener aplicaciones seguras durante su desarrollo.

- Demostrar la importancia que tiene la seguridad en la elaboración de *software* y porque ha alcanzado tanta relevancia en la actualidad.
- Conocer las principales amenazas a las que está expuesto el *software* para entender porque se aplican ciertas metodologías.
- Descubrir las vulnerabilidades de las aplicaciones cuando no se aplican las medidas oportunas.
- Conocer la infraestructura de los principales grupos de aplicaciones para comprender las acciones a realizar en un diseño seguro.
- Presentar diferentes metodologías actuales que se aplican en el desarrollo seguro de *software*.
- Analizar la metodología **DevSecOps** (*Security DevOps*) para demostrar porque es una buena solución para los desarrolladores de *software*.
- Describir las buenas prácticas y recomendaciones que aconsejan los grandes fabricantes de *software* y los profesionales de ciberseguridad en la actualidad.
- Detectar las fallas de seguridad de algunas aplicaciones que *a priori* parecen robustas.
- Presentar las posibles conclusiones obtenidas tras la elaboración del presente trabajo.
- El **objetivo principal** es analizar las mejores técnicas de la metodología DevSecOps y cómo se adapta en el desarrollo de aplicaciones, dependiendo de su arquitectura.

1.5 Enfoque y método seguido

Para desarrollar el presente trabajo, se ha tomado una estrategia de investigación, a pesar de que no se trate de una investigación legítima (como puede ser la elaboración de un nuevo producto o servicio). Se ha tenido que buscar, seleccionar y analizar información de diferentes ámbitos -como libros, *papers* científicos y otras fuentes de Internet-, para poder sentar unas bases y a partir de estas, exponer ideas para obtener diferentes conclusiones.

La metodología utilizada debe ser ágil y cíclica [4], ya que el trabajo se realiza de manera que se van presentando diferentes entregas, pero a medida que se van desarrollando nuevos apartados, los ya creados son susceptibles a nuevos cambios

durante la duración del proyecto. Esta es la mejor metodología que encaja con el trabajo y pueda mejorar poco a poco a lo largo de su desarrollo.

Seguir los tiempos y los plazos para alcanzar los hitos marcados en el diagrama de *Gantt* en la planificación inicial conlleva un gran esfuerzo y dedicación. Sin embargo, es decisivo para "llegar a buen puerto" con el proyecto y vital para obtener buenos resultados.

1.6 Planificación del Trabajo

1.6.1 Tareas a realizar

En la siguiente tabla se muestra la planificación del trabajo que marca los tiempos para realizar las diferentes tareas.

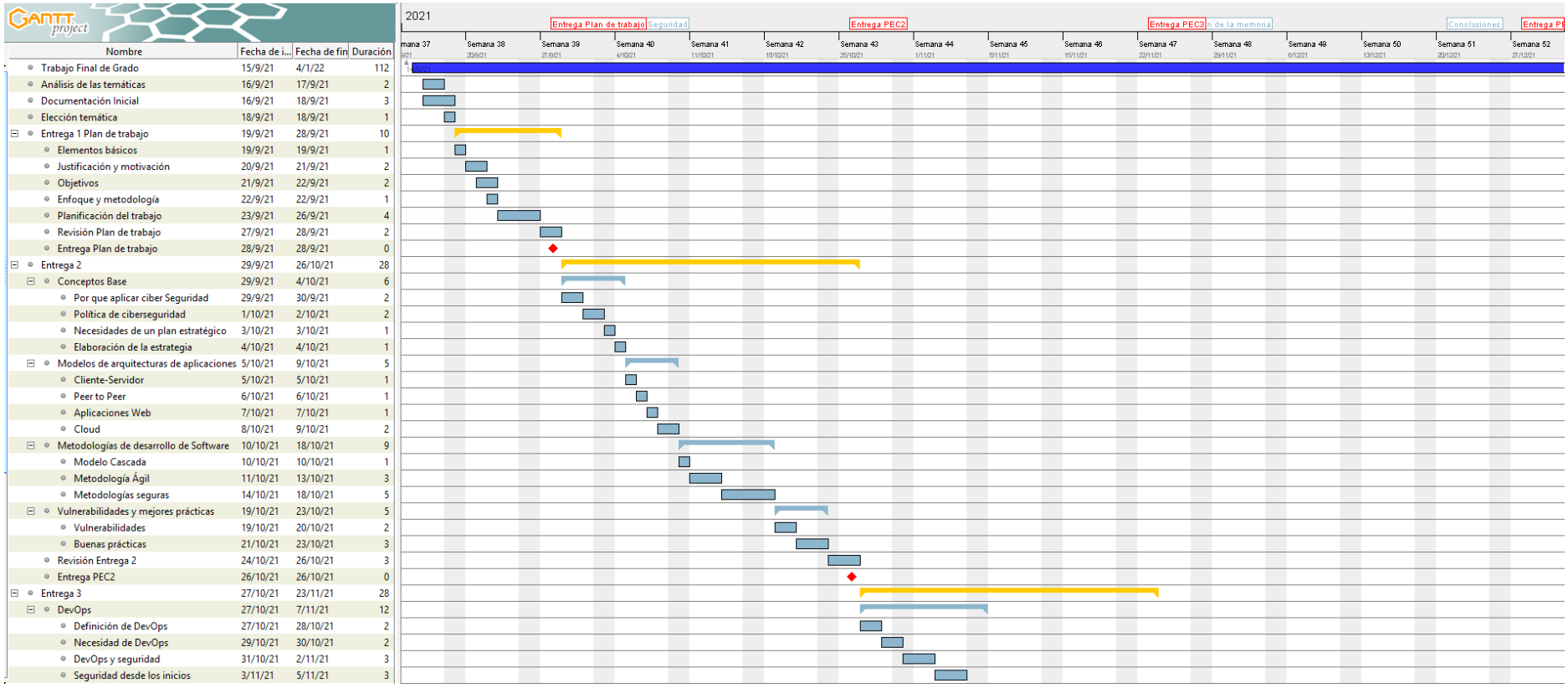
Nombre	Fecha de i...	Fecha de fin	Duración
Trabajo Final de Grado	15/9/21	4/1/22	112
Análisis de las temáticas	16/9/21	17/9/21	2
Documentación Inicial	16/9/21	18/9/21	3
Elección temática	18/9/21	18/9/21	1
Entrega 1 Plan de trabajo	19/9/21	28/9/21	10
Elementos básicos	19/9/21	19/9/21	1
Justificación y motivación	20/9/21	21/9/21	2
Objetivos	21/9/21	22/9/21	2
Enfoque y metodología	22/9/21	22/9/21	1
Planificación del trabajo	23/9/21	26/9/21	4
Revisión Plan de trabajo	27/9/21	28/9/21	2
Entrega Plan de trabajo	28/9/21	28/9/21	0
Entrega 2	29/9/21	26/10/21	28
Conceptos Base	29/9/21	4/10/21	6
Por que aplicar ciber Seguridad	29/9/21	30/9/21	2
Política de ciberseguridad	1/10/21	2/10/21	2
Necesidades de un plan estratégico	3/10/21	3/10/21	1
Elaboración de la estrategia	4/10/21	4/10/21	1
Modelos de arquitecturas de aplicaciones	5/10/21	9/10/21	5
Cliente-Servidor	5/10/21	5/10/21	1
Peer to Peer	6/10/21	6/10/21	1
Aplicaciones Web	7/10/21	7/10/21	1
Cloud	8/10/21	9/10/21	2
Metodologías de desarrollo de Software	10/10/21	18/10/21	9
Modelo Cascada	10/10/21	10/10/21	1
Metodología Ágil	11/10/21	13/10/21	3
Metodologías seguras	14/10/21	18/10/21	5
Vulnerabilidades y mejores prácticas	19/10/21	23/10/21	5
Vulnerabilidades	19/10/21	20/10/21	2
Buenas prácticas	21/10/21	23/10/21	3
Revisión Entrega 2	24/10/21	26/10/21	3
Entrega PEC2	26/10/21	26/10/21	0

Entrega 3	27/10/21	23/11/21	28
DevOps	27/10/21	7/11/21	12
Definición de DevOps	27/10/21	28/10/21	2
Necesidad de DevOps	29/10/21	30/10/21	2
DevOps y seguridad	31/10/21	2/11/21	3
Seguridad desde los inicios	3/11/21	5/11/21	3
Integración y entrega continua	6/11/21	7/11/21	2
DevSevOps	8/11/21	21/11/21	14
Definición DevSecOps	8/11/21	9/11/21	2
Desafíos	10/11/21	11/11/21	2
Características	12/11/21	14/11/21	3
Integración de seguridad	15/11/21	17/11/21	3
Herramientas y seguridad de apps	18/11/21	19/11/21	2
Cadena de suministro segura	20/11/21	21/11/21	2
Revisión Entrega 3	22/11/21	23/11/21	2
Entrega PEC3	23/11/21	23/11/21	0
Entrega 4	24/11/21	28/12/21	35
Memoria Final	24/11/21	28/12/21	35
Maquetación de la memoria	24/11/21	28/12/21	35
Caso práctico	8/12/21	24/12/21	17
Busqueda de información	8/12/21	14/12/21	7
Ejecución	15/12/21	21/12/21	7
Documentación	22/12/21	24/12/21	3
Conclusiones	26/12/21	26/12/21	1
Entrega PEC4	28/12/21	28/12/21	0
Entrega 5	29/12/21	4/1/22	7
Presentación en video	29/12/21	4/1/22	7
Documento presentación y guión	29/12/21	31/12/21	3
Preparación video	1/1/22	2/1/22	2
Grabación Video	3/1/22	4/1/22	2
Entrega Video	4/1/22	4/1/22	0

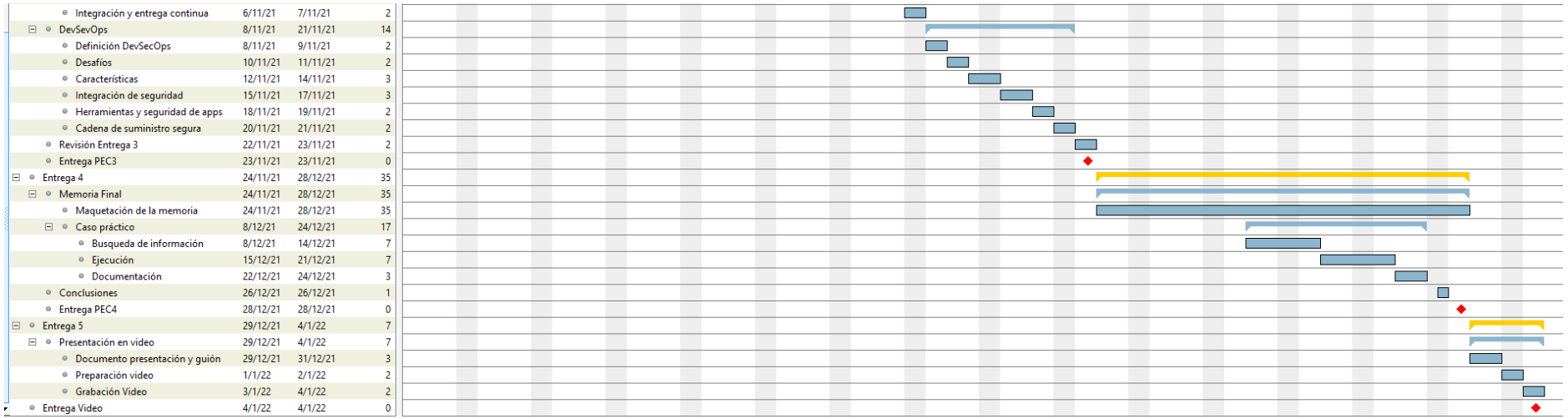
Esquema 1 Planificación

1.6.2 Planificación temporal

La planificación se representa en un **diagrama de *Gantt***, donde se pueden ver las tareas programas desde una vista general.
Las líneas amarillas representan la duración de cada entrega.
Los **hitos**, que en este caso indican las diferentes entregas a realizar durante la vida del proyecto, se representan mediante un rombo rojo.



Nombre	Fecha de i...	Fecha de fin	Duración
Trabajo Final de Grado	15/9/21	4/1/22	112
Análisis de las temáticas	16/9/21	17/9/21	2
Documentación Inicial	16/9/21	18/9/21	3
Elección temática	18/9/21	18/9/21	1
Entrega 1 Plan de trabajo	19/9/21	28/9/21	10
Elementos básicos	19/9/21	19/9/21	1
Justificación y motivación	20/9/21	21/9/21	2
Objetivos	21/9/21	22/9/21	2
Enfoque y metodología	22/9/21	22/9/21	1
Planificación del trabajo	23/9/21	26/9/21	4
Revisión Plan de trabajo	27/9/21	28/9/21	2
Entrega Plan de trabajo	28/9/21	28/9/21	0
Entrega 2	29/9/21	26/10/21	28
Conceptos Base	29/9/21	4/10/21	6
Por que aplicar ciber Seguridad	29/9/21	30/9/21	2
Política de ciberseguridad	1/10/21	2/10/21	2
Necesidades de un plan estratégico	3/10/21	3/10/21	1
Elaboración de la estrategia	4/10/21	4/10/21	1
Modelos de arquitecturas de aplicaciones	5/10/21	9/10/21	5
Cliente-Servidor	5/10/21	5/10/21	1
Peer to Peer	6/10/21	6/10/21	1
Aplicaciones Web	7/10/21	7/10/21	1
Cloud	8/10/21	9/10/21	2
Metodologías de desarrollo de Software	10/10/21	18/10/21	9
Modelo Cascada	10/10/21	10/10/21	1
Metodología Ágil	11/10/21	13/10/21	3
Metodologías seguras	14/10/21	18/10/21	5
Vulnerabilidades y mejores prácticas	19/10/21	23/10/21	5
Vulnerabilidades	19/10/21	20/10/21	2
Buenas prácticas	21/10/21	23/10/21	3
Revisión Entrega 2	24/10/21	26/10/21	3
Entrega PEC2	26/10/21	26/10/21	0
Entrega 3	27/10/21	23/11/21	28
DevOps	27/10/21	7/11/21	12
Definición de DevOps	27/10/21	28/10/21	2
Necesidad de DevOps	29/10/21	30/10/21	2
DevOps y seguridad	31/10/21	2/11/21	3
Seguridad desde los inicios	3/11/21	5/11/21	3



Integración y entrega continua	6/11/21	7/11/21	2
DevSevOps	8/11/21	21/11/21	14
Definición DevSecOps	8/11/21	9/11/21	2
Desafíos	10/11/21	11/11/21	2
Características	12/11/21	14/11/21	3
Integración de seguridad	15/11/21	17/11/21	3
Herramientas y seguridad de apps	18/11/21	19/11/21	2
Cadena de suministro segura	20/11/21	21/11/21	2
Revisión Entrega 3	22/11/21	23/11/21	2
Entrega PEC3	23/11/21	23/11/21	0
Entrega 4	24/11/21	28/12/21	35
Memoria Final	24/11/21	28/12/21	35
Maquetación de la memoria	24/11/21	28/12/21	35
Caso práctico	8/12/21	24/12/21	17
Busqueda de información	8/12/21	14/12/21	7
Ejecución	15/12/21	21/12/21	7
Documentación	22/12/21	24/12/21	3
Conclusiones	26/12/21	26/12/21	1
Entrega PEC4	28/12/21	28/12/21	0
Entrega 5	29/12/21	4/1/22	7
Presentación en video	29/12/21	4/1/22	7
Documento presentación y guión	29/12/21	31/12/21	3
Preparación video	1/1/22	2/1/22	2
Grabación Video	3/1/22	4/1/22	2
Entrega Video	4/1/22	4/1/22	0

Esquema 2 Planificación Diagrama de Grantt

1.6.3 Recursos necesarios

A continuación, se muestran los recursos necesarios para llevar a cabo la planificación mostrada con anterioridad.

Recurso	Necesidad	Coste
Equipo informático ● Sobremesa ○ CPU i5-2500 ○ 8 Gb RAM ○ Disco 500 Gb ● Monitores ○ 22" x 2 ● Teclado ● Ratón ● Cables de conexión ○ Alimentación ○ Ethernet ○ HDMI ● *Router* + ONT	Elemento principal para buscar información y contenido relevante. Uso de diferentes aplicaciones para elaborar el trabajo. Como pueden ser de ofimática, para generar el diagrama de Gantt, entre otras. No es necesario que sea un equipo muy potente, ya que solo se requiere un perfil de ofimática.	Entre 400 y 500 €
Conexión a Internet ● Fibra 300 Mbps	Para realizar las búsquedas necesarias y conectarse al campus. En este caso, se cuenta con una conexión a fibra de 300 Mbps. Sin embargo, se puede realizar sin problemas con un ancho de banda menor.	50 € / mes
Software	**Editor de Texto** Se ha utilizado *Microsoft Word*, pero se puede utilizar cualquier editor gratuito como open office.	Gratis (licencia Universidad)
	Gantt Project Para crear la tabla de planificación y su correspondiente diagrama de Gantt	Gratuito Enlace descarga
	GitHub Desktop Para actualizar los cambios realizados en el proyecto y subirlos a la plataforma web.	Gratuito Enlace descarga
	Visual Studio Code Para editar el código del	Gratuito Enlace descarga

	proyecto.	
	Github Plataforma de colaboración de proyectos.	Gratuito web
	Snyk Herramienta *online* para detectar vulnerabilidades.	Plan gratuito web
	SonarColud Herramienta *online* para revisar la estructura del código.	Plan gratuito web

Tabla 1 Recursos

1.7 Breve sumario de productos obtenidos

Entregables
Se han realizado diferentes entregas que han confeccionado el resultado final del presente trabajo.

- PEC1 --> Plan de trabajo
- PEC2 --> Desarrollo de los temas presentados en la primera entrega
- PEC3 --> Desarrollo de los temas presentados en la primera entrega
- PEC4 --> Desarrollo del caso práctico y entrega de la memoria final
- PEC5 --> Presentación en video del trabajo

Productos conceptuales
Con los primeros capítulos de la memoria, se han obtenido los conceptos previos para contar con la base necesaria, que ayuda a asimilar los temas principales de la memoria.

Conocimiento de cómo funciona la metodología DevOps y como DevSecOps la complementa de manera natural para establecer una filosofía de trabajo, donde la seguridad acompaña la vida del proyecto desde sus inicios hasta el final.

Productos cuantificables (resultados)
Gracias al ejemplo práctico, se ha podido comprobar cómo se puede aplicar seguridad a un proyecto DevOps y ver cómo pueden trabajar diferentes herramientas durante el desarrollo del proyecto.

1.8 Breve descripción de los otros capítulos de la memoria

Capítulo 1 - Introducción
Puede tomarse como una introducción donde se especifica la información relativa al contenido -como puede ser el resumen- e información sobre cómo se ha estructurado el trabajo, tanto en contenido, tareas o tiempo. Gracias a este capítulo, el lector puede hacerse una idea general de la temática del documento, el porqué se ha elegido y sus objetivos. Además, se obtiene una idea general de la situación actual de donde parte el documento.

Capítulo 2 - Conceptos base
En este y los siguientes capítulos, se quiere contextualizar la situación actual, dando a conocer las técnicas utilizadas en el pasado y como se han ido adoptando nuevas formas de trabajo, con la intención de posicionar al lector y otorgarle los conocimientos necesarios para comprender la temática principal del trabajo.

En los apartados de este capítulo, se quiere dar una noción básica de porqué es tan importante en la actualidad emplear la ciberseguridad en todos los sistemas, dispositivos o aplicaciones, acompañado de la necesidad de crear y aplicar un plan estratégico para las compañías.

Capítulo 3 - Arquitecturas de aplicaciones
Se desarrolla de forma breve y concisa las características de algunas importantes arquitecturas clásicas y otras más modernas, que suelen ser habituales en el desarrollo del *software*.

Capítulo 4 - Metodologías de desarrollo de software
Se presenta una metodología clásica para conocer los comienzos y las bases en las que se han basado las metodologías modernas, donde han ido emergiendo para adaptarse a las necesidades que requería el desarrollo del *software*.

También se exponen las ideas generales de las metodologías seguras. Se puede afirmar, que se tratan de metodologías modernas añadiendo la capa de seguridad durante todo el desarrollo. Además, se presentan algunas de las metodologías seguras utilizadas en la actualidad.

Capítulo 5 - Vulnerabilidades y mejores prácticas
Una vez conocidas algunas metodologías, se indican algunas de las vulnerabilidades más comunes que sufre el *software* y que lleva arrastrando desde hace décadas. También se comentan algunas buenas prácticas para intentar paliar estas debilidades.

Capitulo 6 - DevOps
DevSecOps se basa en DevOps (*Development* y *Operations*) que define la simbiosis entre las personas que forman los equipos de desarrollo, la tecnología subyacente y los procesos para proporcionar a los clientes un producto de calidad de forma constante.

En este capítulo se exponen los contenidos más relevantes que proporcionan la base para DevSecOps. Es importante extender una cultura de trabajo que mejora el rendimiento a la vez que reduce el tiempo de desarrollo, promoviendo la colaboración y la productividad.

Capitulo 7 - DevSecOps
DevSecOps añade la capa de seguridad a DevOps, agregando prácticas seguras durante el proceso de desarrollo y entrega de *software* de manera colaborativa. Resuelve el dilema que se produce entre lanzar lo más rápido posible las versiones de *software* por parte de los equipos DevOps y la prioridad de la seguridad por los equipos de seguridad.

Al integrar las técnicas y las buenas prácticas de seguridad en el ciclo del desarrollo de *software*, este puede ser entregado de manera ágil a la vez que es robusto ante posibles vulnerabilidades de seguridad.

Capítulo 8 - Caso Práctico
La teoría se lleva a la práctica. Se toma un proyecto de un repositorio público al cual, se le modifica el código para que genere una vulnerabilidad reciente. A continuación, se configuran dos herramientas para que realicen escaneos y test en busca de vulnerabilidades o fallos en el código.

2. Conceptos base

2.1 Por qué aplicar ciberseguridad

Desde finales del siglo pasado, -donde la documentación de empresas y sedes gubernamentales se encontraba en formato papel y almacenaba en grandes salas- la digitalización de la información comenzó tímidamente, pero con el pasar de los años, se ha convertido en una situación normalizada, tanto en el área laboral como personal. El irrumpir de las **TIC** (Tecnologías de la Información y comunicación [12]) ha marcado el comienzo de un nuevo estilo de trabajo.

Este cambio, provocó el desarrollo de nuevas infraestructuras, *software* o *hardware*, donde las comunicaciones han tomado un papel protagonista para interconectar el mundo actual en el que vivimos. Como objetivo primordial, disponer de la información que se necesite en el momento que sea necesario. Pero exponer esta información tiene tantos peligros que es necesario aplicar diferentes políticas para asegurar la tecnología [13].

2.2 Política de ciberseguridad

Se puede definir como un conjunto de normativas donde se recogen las buenas prácticas, procedimientos o planes estratégicos que marcan el camino de una compañía a la hora proteger sus datos y otros diferentes elementos. Debido al constante cambio de las tecnologías, estas políticas no deben ser rígidas, y han de evolucionar a lo largo del tiempo, a partir de las experiencias vividas junto al crecimiento que sufre la empresa.

Las políticas de seguridad marcan como su objetivo principal, conservar la integridad, la confidencialidad y la disponibilidad de la información. Sin embargo, no solo con estas medidas es suficiente para proteger los bienes de la empresa. También se debe concienciar y educar a los empleados que conviven con estas normas, ya que pueden entenderlas como restricciones que impiden realizar su trabajo en lugar de medidas de protección del mismo. Deben entender por qué se aplican y cuáles son sus obligaciones.

2.3 Necesidad de un plan estratégico de seguridad

Dependiendo del tipo de organización y las aplicaciones que vaya a utilizar, entran en juego un sinfín de variables para determinar qué normas de seguridad se quieren utilizar para la arquitectura, *hardware* o *software*. Algunas de las más habituales pueden ser:

- Dependientes de la arquitectura de red con la que debe conectar.
- *Software* malicioso que corrompa el sistema.

- Aumento exponencial del cibercrimen como forma de negocio

Aplicando estas y otras normativas dentro del método de trabajo se pueden obtener diferentes beneficios.

- Fomenta la toma de mejores decisiones, ya que adquieren un papel relevante en la toma de decisiones para alcanzar los objetivos marcados.
- Mejora de la seguridad. Se satisfacen los requisitos para que los procesos cumplan con la seguridad y calidad que exige el mercado, donde empleados y clientes obtienen beneficios.
- Protege la productividad. Toda la información se mantiene segura y fuera del alcance de personas no autorizadas.
- Muestra robustez y confianza ante los clientes, ya que tienen la tranquilidad de que los datos son gestionados de manera segura.

2.4 Elaboración de la estrategia correcta

Para elaborar el conjunto de políticas adecuadas, se debe analizar todo el proceso de producción e interpretar cómo los empleados (los diseñadores en el caso de *software*) interaccionan en el desarrollo. Se deberían localizar objetivos que se quieren proteger y a partir de aquí, elegir entre las siguientes vertientes.

- Prevención. Formada por acciones como el control de acceso, autenticación, seguridad de comunicaciones...
- Detección. Descubrir los intentos o infracciones de las normativas de seguridad.
- Recuperación. Una vez detectada y asimilada la infracción, aplicar medidas para restaurar la situación previa.

Es importante contar con personas preparadas en la seguridad en los diferentes campos que requiera la organización, capaces de implantar las directrices que debe seguir todo el equipo durante el desarrollo. Para que esto ocurra, es importante familiarizarse con una cultura de seguridad hasta que se normalice como estilo de trabajo [14].

3. Arquitecturas de aplicaciones

Como se ha comentado anteriormente, existen diferentes tipos de arquitecturas en las que se basa el desarrollo de cada aplicación para adaptarlas a las necesidades de los clientes o usuarios finales que vayan a trabajar con las mismas. A continuación, se expone un breve resumen de algunas de las arquitecturas más extendidas.

3.1 Arquitectura Cliente-servidor

Se caracteriza por contar con un número indeterminado de *hosts* cliente, que se intercomunican a través de la red para solicitar un cierto servicio, el cual es proporcionado por uno o varios servidores centralizados. A los clientes se les facilita un interfaz para que puedan interactuar con el servicio ofrecido [15].

Un ejemplo muy habitual, es donde los clientes realizan diferentes acciones con una aplicación, como modificar o ingresar nueva información, estos envían los cambios realizados al servidor, que es quien almacena una Base de Datos y el encargado de correr el gestor que manipula y modifica los cambios realizados por los clientes.

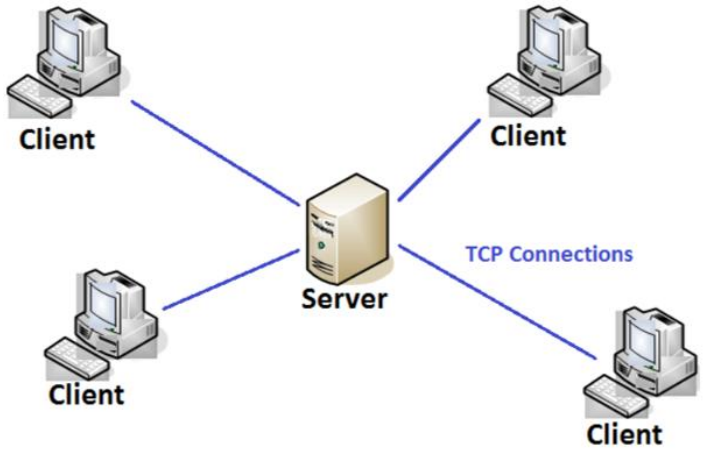


Figura 1 Arquitectura cliente-servidor

También es habitual ofrecer escritorios virtuales, donde los clientes no requieren grandes características *hardware* o *software* (este papel lo asume el servidor, ya que es el encargado de realizar todo el procesamiento), con una configuración básica y una red que capacite a acceder al servicio es suficiente.

Los llamados *dumb terminals* (equipos tontos) solicitan el escritorio al servidor y este se lo proporciona. Todas las tareas de procesamiento y carga de memoria realmente se ejecutan en el servidor, simulando que ocurre en el cliente, cuando realmente solo le está llegando la imagen de lo que ocurre en el servidor.

Sus mayores desventajas son la sobrecarga de los servidores, (esto sucede cuando se realizan más peticiones de manera simultánea de las que el servidor es capaz de atender) y al mantener el servicio de manera centralizada, la falla de algún servidor puede provocar la caída del servicio, dando como resultado, una pérdida de producción o monetaria.

3.2 Arquitectura *Peer to peer*

En comparación con la arquitectura cliente-servidor, se elimina el papel de servidor, por lo tanto, se rompe la centralización del servicio enfocado a unas pocas máquinas. Aquí, cada dispositivo actúa con ambos perfiles dependiendo de las necesidades de cada momento, es decir, cada *host* es capaz de enviar sus propias peticiones y atender las de otros *hosts*. Algunos ejemplos son, aplicaciones como *Skype* o *Bit Torrent*.

Su mayor virtud, es la gran escalabilidad que puede producir en poco tiempo y con pocos recursos, ya que solo es necesario añadir host a la red para que comiencen a participar de forma colaborativa. Puede padecer contradictorio ya que, en otras arquitecturas, contra más clientes solicitan servicios, más se saturará el sistema y más recursos serán necesarios.

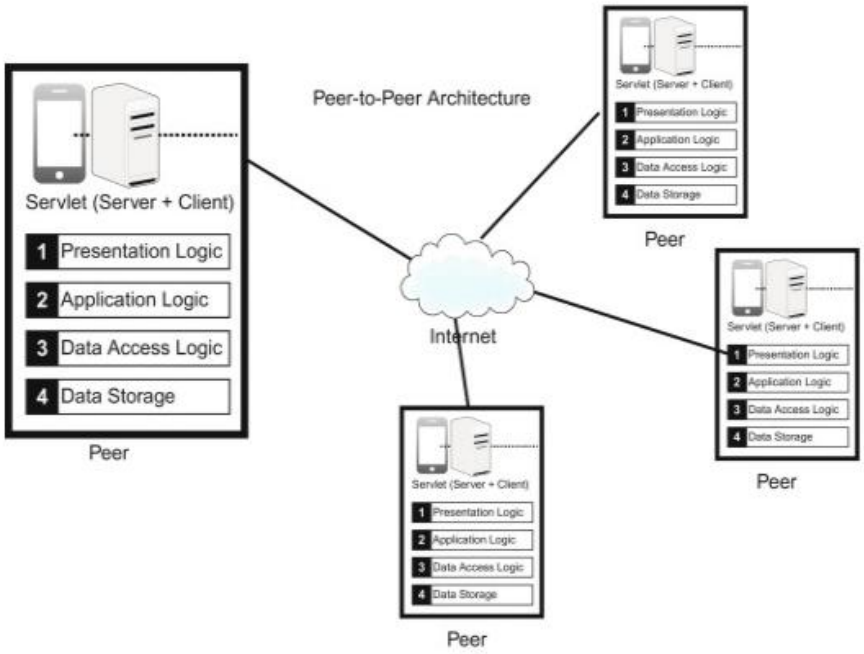


Figura 2 Arquitectura *Peer to peer*

La contra de este sistema es que los desarrolladores de *software* pierden gran parte o incluso todo el control sobre los usuarios que utilizan la aplicación. Esto reduce considerablemente la elección de esta arquitectura a un tipo muy determinado de aplicaciones. Existen modelos híbridos que cuentan con servidores centrales, pero manteniendo la idea de servicio descentralizado [16].

3.3 Aplicaciones Web

Las aplicaciones se pueden definir como programas a los que se debe acceder a través de un navegador web con acceso a Internet o una red local (Intranet) para poder operar con sus funcionalidades. Todas las *webs* de *e-commerce* (como Amazon), blogs, foros o las páginas de Bancos se basan en esta arquitectura. También proporcionan la

alternativa a instalar *software* de manera local en cualquier equipo, ofreciendo las mismas funcionalidades en su versión web (como paquetes de ofimática o clientes de correo).

Es una variante de la ingeniería del *software,* ya que se debe diseñar de tal manera que se ejecute en servidores web, en lugar de manera local en una máquina cliente. Como en muchas ocasiones hay un negocio detrás de la web, esta debe ser atractiva para los usuarios. Por lo que el desarrollador en muchas ocasiones debe contar con experiencia en diferentes campos, como *marketing* o diseño gráfico.

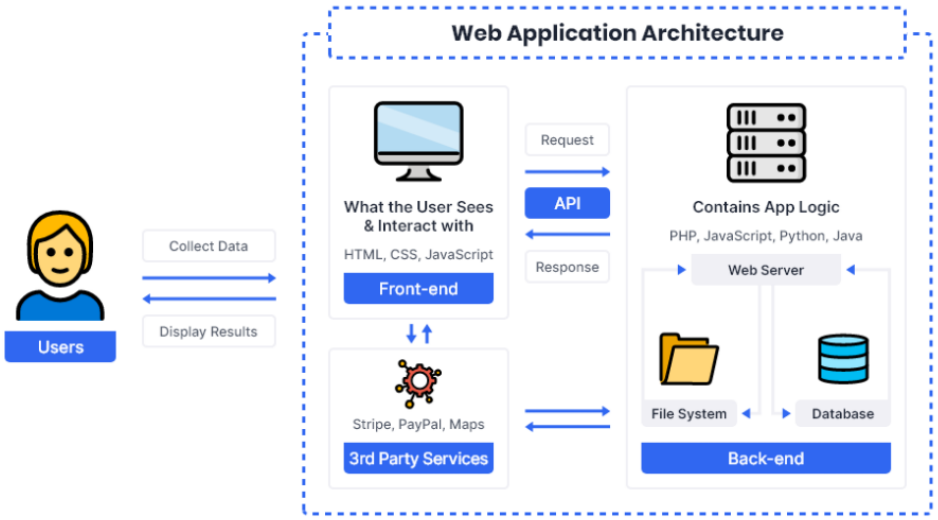


Figura 3 Arquitectura aplicación web

Los servidores web, pueden ofrecer el servicio basándose en una arquitectura cliente servidor (vista con anterioridad) o también ofreciendo el servicio entre servidores múltiples. Es decir, un servidor recibe la petición del cliente web, la cual analiza e identifica que los diferentes objetos solicitados están contenidos en diferentes servidores. El conjunto de servidores se comunican entre sí para que cada uno aporte los objetos que apodera para enviarlos y satisfacer la solicitud del cliente [17].

3.4 Arquitectura en capas (N-tier)

Se suele asociar con aplicaciones heredadas y por lo general, se utiliza para diseñar aplicaciones empresariales. Suelen estar formadas por varias capas (tres o más) que modelan los componentes de la aplicación y cada una de ellas cumple una función particular.

Las capas gestionan las dependencias y las funciones lógicas. Estas se organizan de forma horizontal donde solo pueden acceder a las funcionalidades de capas inferiores, es decir, solo pueden utilizar las funciones de la capa situada justo debajo o las inferiores a esta.

3.5 Arquitectura monolítica

Son pilas de aplicaciones que contienen todas sus funcionalidades y cuentan con conexión directa entre los servicios, junto a los procesos de desarrollo y distribución. Esto quiere decir que, al actualizar alguna funcionalidad de la aplicación, afectará además a toda la infraestructura.

Las actualizaciones y nuevas versiones se suelen dar pocas veces al año (una o dos) ya que implica lanzarla de manera completa. En contraste, las arquitecturas más actuales intentan realizar lanzamientos priorizando la agilidad.

3.6 Arquitectura de microservicios

También se trata de un modo de escribir código. Las aplicaciones se dividen en elementos independientes más pequeños, denominados microservicio. Ofrece una escalabilidad dinámica como la tolerancia a fallos, donde los servicios pueden ser ampliados según necesidad.

Su objetivo es distribuir un *software* de calidad en menor tiempo. Al poder desarrollar múltiples multiservicios de forma simultánea, los desarrolladores pueden trabajar en sus servicios al mismo tiempo, en lugar de tener que actualizar toda la aplicación.

Con los equipos DevOps y APIs, los microservicios de contenedores forman la base de las aplicaciones nativas en la nube.

3.7 Arquitectura basada en eventos

Los eventos son los acontecimientos importantes que experimenta tanto el *hardware* como el *software*. Esta arquitectura realiza un acoplamiento mínimo, por lo que se adapta muy bien a aplicaciones distribuidas.

Se constituye por productores –que detectan los eventos y los transmite- y consumidores -que los reciben a través de los canales de procesamiento asíncrono-. Se basan en un modelo de publicación/subscripción o flujo de eventos.

Una vez que se publica un evento, se envía a la lista de suscriptores que deben ser informados. Estos eventos se capturan cuando se generan desde sus fuentes –como redes, dispositivos IOT o aplicaciones- donde los productores y consumidores comparten información del estado y respuestas en tiempo real [18].

3.8 Cloud

Dependiendo de cómo se analice, quizás no se pueda considerar una arquitectura, sino más bien una plataforma de despliegue. El servicio *cloud*, combina diferentes tecnologías para formar una nube de recursos, donde estos se agrupan con la ayuda de la virtualización y son compartidos a través de la red [19]. Está formada por los siguientes componentes:

- Plataforma *front-end*. Utilizada por el cliente o dispositivo final para acceder a la nube.
- Plataformas *back-end*. Conforma el conjunto de servidores y almacenamiento donde se ejecuta el servicio.
- Tecnología de entrega. Dando respuesta basada en la nube a las solicitudes realizadas por los clientes.
- Una red. Con la que permita conectar tanto a los servidores y almacenamiento con los clientes.

Esta combinación de tecnologías genera una computación en la nube donde diferentes aplicaciones pueden ejecutarse, ofreciendo a los clientes la potencia de todos estos recursos. Para las empresas que adoptan esta arquitectura, tienen la ventaja de liberarse tanto de servidores como el almacenamiento en modo local (*on-premise*), reduciendo considerablemente los elementos que forman sus centros de datos (también ahorran en elementos secundarios como luz, aire acondicionado, etc.). La arquitectura *cloud* se puede dividir en tres modelos:

- **Software como Servicio** (SaaS). Los proveedores del servicio son los encargados del mantenimiento y la entrega de *software* al cliente a través de Internet, evitando que los usuarios finales tengan que ejecutar ninguna aplicación de manera local en sus equipos. Es habitual que las aplicaciones se entreguen a través de una interfaz web para que sea accesible desde cualquier dispositivo conectado a la red.

- **Plataforma de Servicio** (Paas). El proveedor de servicios ofrece una plataforma completa adaptada a sus necesidades donde en muchas ocasiones también se ofrece un *middleware* (*software* intermedio). Es decir, pone a disposición del cliente los servidores, almacenamiento y las redes para alojar su servicio o aplicación requerida. Gracias a esta infraestructura en la nube, las compañías tienen la libertad de generar aplicaciones o servicios, por lo que solo se tienen que preocupar de la implantación del *software* y su configuración.

- **Infraestructura como servicio** (IaaS). Se puede denominar como el formato más sencillo de arquitectura en la nube. El proveedor ofrece la infraestructura (servidores, almacenamiento y red) al cliente, que se encarga de la gestión de sus aplicaciones y suelen pagar de manera mensual la capacidad de los recursos necesarios que va requiriendo.

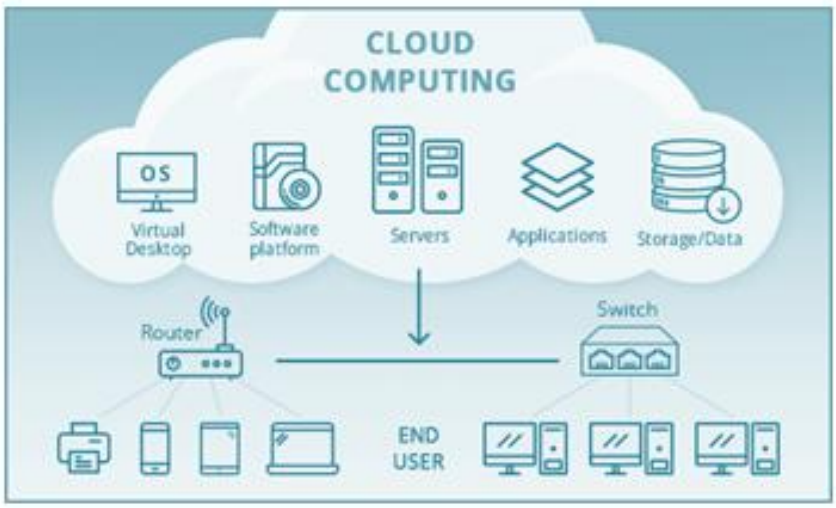


Figura 4 Arquitectura Cloud

3.9 Análisis de seguridad de arquitecturas

Los equipos con los aplicativos instalados de manera local, como en la arquitectura cliente-servidor, suelen ser potenciales vectores de ataque a ojos de un ciber delincuente, como cambios de permisos en BBDD de SQL, debilidades *backends* de los servicios web y de las APIs o modificar los archivos de configuración, entre otros. Es cierto que se pueden tomar medidas para paliarlos, pero el riesgo siempre está presente [20].

Las soluciones SAS ofrecen mejoras respecto a las arquitecturas locales respecto a servicios, como mantenimiento, disponibilidad y seguridad. En caso de sufrir un ataque o cualquier tipo de falla que deje inoperativo el sistema, contar con un servicio *cloud*, asegura su operatividad de manera casi inmediata reduciendo el impacto de los usuarios y de la compañía.

La arquitectura de microservicios ha desbancado a las iniciales estructuras monolíticas en el despliegue de aplicaciones *cloud* (poco eficientes con alta concurrencia de usuarios), que no contaban con la separación de los módulos que forman el *software*. Gracias a esta descentralización, se consigue afrontar cualquier incidencia de forma independiente sin afectar al resto de elementos que forman el *software* [21].

4. Metodologías de desarrollo de *software*

Existen multitud de metodologías a la hora de desarrollar *software*, algunas más actuales y otras más antiguas que se llevan arrastrando a lo largo del tiempo. Se va a proceder a exponer las características más relevantes de dos tipos de metodologías (modelo en cascada o ciclo de vida y la metodología ágil) para situar al lector en la situación actual, la evolución que han sufrido las técnicas de desarrollo y cómo han cambiado sus principales ideales. **Metodología** se puede definir como "*conjunto de métodos coherentes y relacionados por unos principios comunes* [22]".

4.1 Modelo en cascada

El modelo en cascada se puede englobar dentro del conjunto de metodologías clásicas, donde el objetivo de su implantación era ordenar el caos existente que reinaba durante el comienzo masivo de la generación de *software*, allá por la década de 1960. Otros tipos de metodologías clásicas pueden ser la incremental, espiral o de prototipos [23].

Presenta un enfoque poco iterativo y por lo tanto poco flexible. El motivo principal es que desde el inicio del proyecto se establecen las diferentes etapas, las cuales están poco adaptadas a cambios durante el desarrollo del proyecto. Para algunos proyectos puede encajar bien, ya que es muy fácil de implementar, como los que están bien definidos desde el comienzo y esperan pocas variaciones.

Es verdad que es un modelo muy básico, pero ha establecido las bases de nuevos paradigmas que han evolucionado a partir de este. Se basa en la implantación de diferentes etapas bien definidas, por las que tiene que pasar el desarrollo de manera secuencial y lineal, donde el inicio de cada fase depende de los resultados de la anterior. Cada etapa está formada por un conjunto de actividades e hitos que se deben lograr para alcanzar las metas requeridas.

Estas etapas se suelen definir como: análisis, diseño, implementación, pruebas y despliegue.

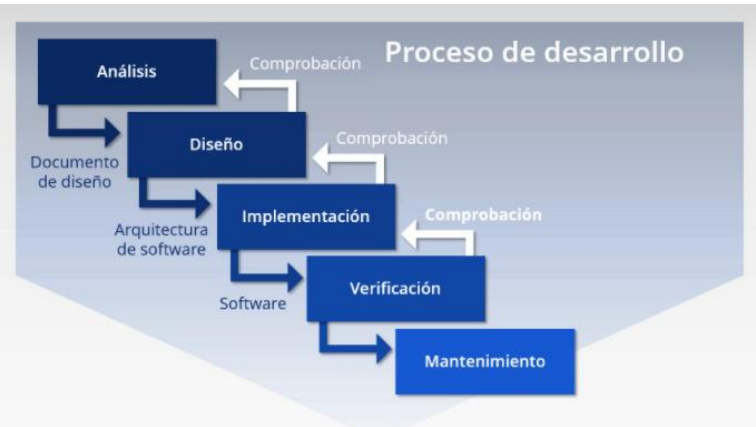


Figura 5 Modelo en cascada de Winston W. Royce

Las contraindicaciones que han aparecido a lo largo del tiempo con la evolución del desarrollo de nuevo *software* han provocado que esta metodología caiga en desuso. Algunas desventajas son:

- Dificultad para introducir cambios cuando una etapa ha concluido.
- Los requisitos iniciales son cruciales para el resto del proyecto.
- Al no contar con retroalimentación, cualquier problema que se presente durante la vida del proyecto, se arrastra durante el resto del mismo y son difíciles de solucionar, lo que provoca grandes retrasos.
- Solucionar cualquier percance conlleva gran cantidad de tiempo y recursos [25].

4.2 Metodología ágil

En el desarrollo de *software* desde un planteamiento ágil, se marca como objetivo distribuir aplicaciones mediante diseños formados con iteraciones rápidas. Así mismo, no marca una serie rígida de estepas como en el método en cascada, más bien, es una forma de pensar, para obtener un proceso colaborativo mediante flujos de trabajo (*workflows*), para dar respuestas ante los cambios.

Este enfoque, nace a principios del nuevo milenio cuando numerosos desarrolladores insistían en que el planteamiento en cascada estaba obsoleto, ya que no satisfacía las necesidades de los clientes y tampoco proporcionaban los resultados que esperaban. La pérdida de recursos y tiempo condujo a que se buscara una alternativa. La transformación digital era imparable y era necesario cambiar el modelo.

Busca generar en periodos cortos de tiempo, pequeñas versiones del *software* donde no están disponibles todas las funcionalidades de la aplicación. Sin embargo, son de gran utilidad, ya que cuando se entrega al cliente, se aprende y modifica el proyecto según sus impresiones o *feedback*. Así, es posible realizar **mejoras de manera cíclica** mientras se mantiene un contacto estrecho entre desarrolladores y el cliente durante el ciclo de vida del proyecto.

Muchos flujos de trabajo en la actualidad se rigen en desarrollos ágiles, como puede ser el *cloud computing*, con una infraestructura muy escalable. El desarrollo en la nube toma el enfoque ágil ya que ofrece servicios que interactúan entre sí para satisfacer las necesidades de los clientes.

Los ámbitos ágiles de desarrollo –como Scrum- son la base de procesos más conocidos como **DevOps**. Con una mentalidad de trabajo que divide a los participantes en pequeños grupos (de 5 a 9 personas), la labor se divide en pequeñas tareas que se definen como *sprint*. En pocas semanas se producen ciclos de retroalimentación en forma de reuniones con la intención de mejorar en cada ciclo.

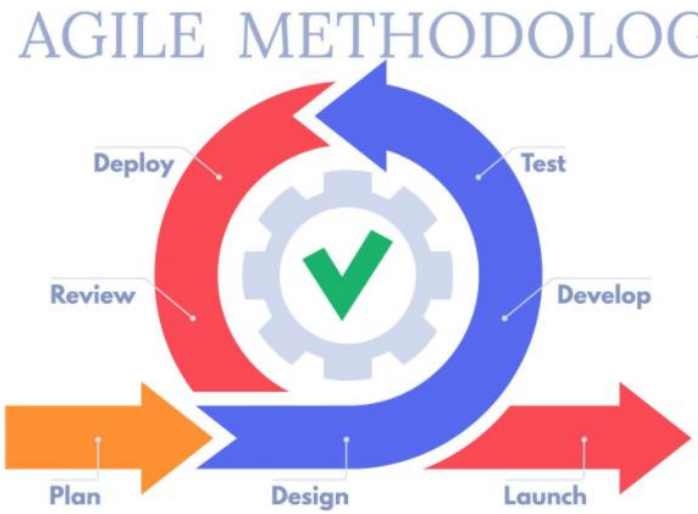


Figura 6 Modelo ágil de desarrollo

4.3 Metodologías seguras

A continuación, se exponen tres ejemplos de los más populares entre las metodologías actuales basadas en el método ágil, donde se añade la capa de seguridad durante el desarrollo del proyecto.

4.3.1 CLASP (*Comprehensive Lightweight Application Security Process*)

CLASP es un conjunto de componentes y actividades inspirado en prácticas que se han formalizado basadas en la experiencia. Su función es facilitar la tarea a los equipos de desarrollo para incorporar la capa de seguridad durante las primeras etapas del ciclo de vida del desarrollo *software* de manera estructurada.

La compañía *Secure Software* [26] ha desarrollado esta metodología en base de la experiencia de sus empleados, donde el proceso se descompone en diferentes ciclos de vida con el objetivo de obtener diferentes requisitos de seguridad. Son estos requisitos los que hacen de base de la metodología CLASP. Estas buenas prácticas permiten a las compañías seguir un guión con el que poder abordar las posibles vulnerabilidades para impedir que sean explotadas. Algunas son, por ejemplo, confidencialidad, autenticación y autorización.

El proceso CLASP se divide en cinco etapas llamadas **vistas CLASP** (CLASP *Views*). Permite a los desarrolladores entender de manera sencilla el proceso, cómo interactúan los componentes y cómo aplicarlo a sus desarrollos [29].

1. **Vista de conceptos**. Proporciona una introducción de alto nivel a la metodología. Describe de manera breve la interacción de las diferentes vistas, las mejores prácticas, en que secuencia aplicar los componentes…

2. **Vista basada en roles**. Contiene instrucciones del proceso basadas en funciones.
3. **Vista de evaluación de actividades**. Ayuda a los responsables de proyecto a evaluar las actividades más idóneas para aplicar durante el desarrollo. CLASP proporciona cierto soporte para ayudar a elegirlas.
4. **Vista de implementación de actividades**. Contiene todas las actividades CLASP de seguridad que pueden ser integradas en el desarrollo del *software*. Las actividades de SDLC generan un *software* seguro gracias a las actividades evaluadas y aceptadas durante la evaluación.
5. **Vista de vulnerabilidades**. Está formada por un catálogo de diferentes problemas identificados previamente por CLASP que forman la BBDD de vulnerabilidades que se generan en el código fuente del *software*. Estas se clasifican en cinco categorías diferentes. Además, asociados a esta vista, se describen las condiciones en las que los servicios de seguridad suelen tener vulnerabilidades en la capa de aplicación. También se proporcionan ejemplos prácticos muy sencillos de entender relacionándolos con el origen de la falla y sus posibles soluciones.

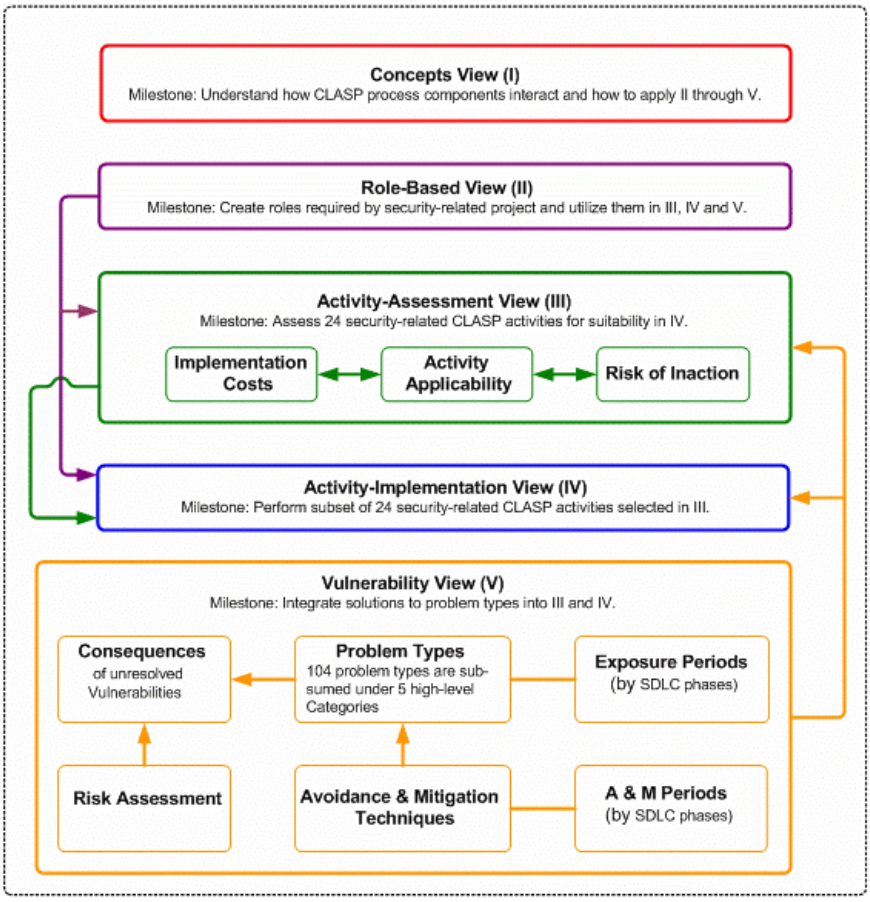


Figura 7 Vistas CLASP

4.3.2 SSDF (*Secure Software Development Framework*)

Es un conjunto de buenas prácticas seguras y consolidadas. Se basan en prácticas de desarrollo de *software* seguro establecidas por organizaciones como BSA, OWASP o *SAFECode*. No son muchos los modelos de ciclo de vida que abordan la seguridad durante el desarrollo del *software*, por lo que las actividades de SSDF deben integrarse en cada implementación de cada ciclo SDLC (*Software Development Life Cycle*).

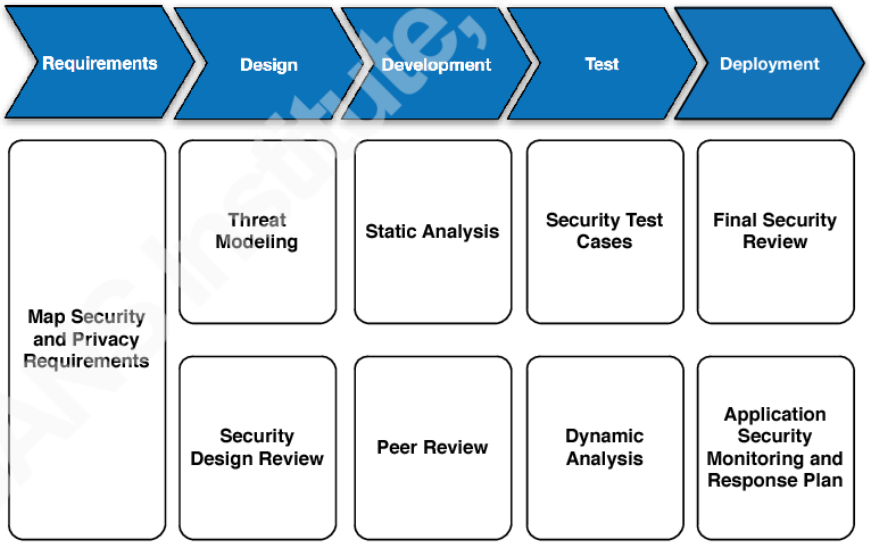


Figura 8 Principios y prácticas de SSDF

Algunas de sus prácticas son [30]:

- Definir criterios para las comprobaciones de seguridad del *software*.
- Proteger todas las formas de código contra el acceso no autorizado y la manipulación, salvaguardando los entornos de desarrollo, construcción, distribución y actualización y siguiendo el principio de mínimo privilegio.
- Proporcionar un mecanismo para verificar la integridad de la versión del *software* mediante la firma digital del código a lo largo del ciclo de vida del *software*.
- Diseñar el *software* para cumplir con los requisitos de seguridad y mitigar los riesgos de seguridad.
- Verificar que el *software* de terceros cumple con los requisitos de seguridad.
- Configurar los procesos de compilación y construcción para mejorar la seguridad de los ejecutables.
- Revisar y/o analizar el código legible por el ser humano para identificar vulnerabilidades y verificar el cumplimiento de los requisitos de seguridad.
- Probar el código ejecutable para identificar vulnerabilidades y verificar el cumplimiento de los requisitos de seguridad.
- Configurar el *software* para que tenga una configuración segura por defecto.
- Archivar y proteger cada versión del *software*.
- Identificar, analizar y corregir las vulnerabilidades de forma continua.

4.3.3 SSDL (Secure Software Development LifeCycle)

Con la abrupta aparición en escena del IOT (*Internet Of Things*) las amenazas no quedan limitadas a los equipos habituales, sino que se extienden a otros elementos como coches o aparatos médicos, al mismo tiempo que las grandes empresas no son el único blanco de estas amenazas. Por estos y otros motivos, se debe implementar la seguridad durante el ciclo de vida del desarrollo del *software* [31].

Hay un problema recurrente y es considerar a los desarrolladores como el eslabón más débil a la hora de aplicar seguridad a las aplicaciones. Aunque tienen nociones sobre seguridad, no se les puede confundir con expertos en este campo.

Hay una solución que ofrece un enfoque para la seguridad de aplicaciones que se aplica de manera estructurada, se trata del SSDL (*Secure Software Development Lifecycle*). Describe un conjunto de acciones que se integran durante todas las etapas de desarrollo y posterior mantenimiento de las aplicaciones. Se puede definir como un método ágil añadiendo una capa de seguridad.

Ofrece muchas ventajas, entre las que destacan:

- **Aumento de la seguridad**. Aplica una supervisión continua de las posibles vulnerabilidades para mejorar la calidad del *software* y reducir los riesgos.
- **Reducción de costes**. Al solventar los posibles fallos, se reduce de manera considerable la detección a *posteriori* y sus soluciones.
- **Cumplimiento de las normativas**. Fomenta que los desarrolladores tomen una actitud concienzuda para cumplir con la normativa y reglas de seguridad. Ignorarlas puede conllevar sanciones económicas.
- **Otros beneficios** pueden ser la formación continua de los equipos de desarrollo, enfoques de seguridad más coherentes, clientes más confiados y con mayor tranquilidad…

Es conveniente especificar las etapas que se van a aplicar para reforzar la seguridad para conocer de manera clara, qué y cuándo se debe hacer. Las etapas de desarrollo y recomendaciones comunes suelen ser las siguientes [31].

Figura 9 Desarrollo ágil SDL

1. **Concepto y planificación.**

Se define la finalidad de la aplicación, incluyendo el plan del proyecto completo, los requisitos y asignación de recursos.

Descubrimiento, definición de objetivos, selección de la metodología SDL y el plan detallado de las actividades.

Se listan los requisitos de seguridad que requiere la aplicación.

Se realizan sesiones de formación para que los equipos cuenten con los conocimientos adecuados que requiere el proyecto.

2. **Arquitectura y diseño**

El objetivo es diseñar un producto que cumpla los requisitos, además de incluir los escenarios de uso junto a los componentes de terceros que puedan ser requeridos.

Se identifican las posibles amenazas para añadir funcionalidades a la aplicación y así estar preparada si sufriera alguna. A continuación, se documenta el diseño que se valida contrastándolo con los requisitos de seguridad. Finalmente se analiza el *software* de terceros para mantenerlo siempre actualizado, incluso sustituirlo si la situación lo requiere.

3. **Implementación**

Se forma con la creación del código, depuración y el lanzamiento de versiones estables para las pruebas.

Las guías sobre la codificación segura apoyan a los programadores con la intención de evitar los clásicos errores que producen vulnerabilidades. Aplicaciones de escaneo de código (SAST) localizan sus posibles debilidades, aunque siempre se realiza la comprobación manual a *posteriori*.

4. **Prueba de corrección de errores**

   Realiza pruebas automáticas y manuales en busca de posibles problemas.

   Herramientas de escaneo automático (DAST) simulan ataques en tiempo de ejecución. Las pruebas de f*uzzing* mejoran la protección contra ataques, al generar entradas aleatorias basadas en diferentes patrones.

5. **Liberación y mantenimiento**

   En esta fase se dispone de muchas instancias del *software* corriendo en entornos paralelos. A cada una de ellas se le aplican diferentes actualizaciones o parches.

   La supervisión debe abarcar todo el entorno del sistema y no solo el de la aplicación. También se crea un plan de respuesta en caso de accidente, donde se indican los procedimientos a seguir si se produce alguna falla de seguridad. Además, se realizan comprobaciones de vulnerabilidades de manera cíclica para responder ante amenazas emergentes.

6. **Fin de vida**

   Los desarrolladores dan por finalizado su trabajo y se desvinculan del proyecto.

   Se deben cumplir las normativas sobre la retención de datos que dictaminan los gobiernos para cumplir los requisitos legales por parte de la empresa. Al final de la vida útil del *software*, los datos sensibles deben de ser eliminados cuidadosamente, como claves de cifrado o datos personales para evitar las posibles filtraciones [32].

5. Vulnerabilidades y mejores prácticas

5.1 Vulnerabilidades en el desarrollo de software

Recientemente *Mitre Corp* [33] (Centro de investigación de EEUU que cuenta con numerosos laboratorios de ciencia y tecnología con un presupuesto de varios miles de millones de dólares [34]) ha publicado los errores más comunes y peligrosos que se suelen cometer en la codificación de *software*, [35] –algunos de ellos llevan años en la lista-.

Se hace difícil comprender en un principio como a pesar de las recomendaciones que se repiten todos los años, muchos desarrolladores no tengan interiorizado el proceso de aplicar seguridad a sus aplicaciones. Sin embargo, detrás existen intereses y negocios que, en muchas ocasiones, chocan con una mentalidad segura, y se interpreta como una barrera que coloca obstáculos en el camino en lugar de una necesidad lógica.

A primera vista, se otorga mayor importancia a solucionar el funcionamiento de los algoritmos que no realizan sus tareas como se espera. Si alguna restricción de seguridad puede estar interponiéndose, se decide ir por la vía sencilla y desactivar estas restricciones. Ya que estas acciones quedan ocultas hasta que se explota la vulnerabilidad.

Los probadores (*testers*) tienen la gran responsabilidad de garantizar la seguridad en el *software*. Sin embargo, en muchas ocasiones no cuentan ni con los conocimientos ni las herramientas adecuadas. También se suele dar por hecho, que esta responsabilidad recae en el equipo de producción IT, ya que cuentan con herramientas para gestionar y securizar las redes con *firewalls* y *antimalware*.

A continuación, se exponen clásicas vulnerabilidades que se llevan explotando durante décadas, y a pesar de ser una situación conocida, siguen siendo elementos confiables y efectivos para muchos atacantes a nivel mundial [36].

Desbordamiento de memoria o buffer (Buffer/Memory Overruns)
Es un ataque muy popular. Si el atacante tiene acceso a una dirección de memoria específica (utilizada por un *software*), puede introducir valores o funciones que sobrepasan su capacidad. En este momento, se inserta un *software* atacante con el objetivo de controlar el equipo o cambiar ciertos permisos de acceso. Suele ser común que suceda cuando los programadores no han limitado el tamaño de las variables que utiliza el programa.

Cross-Site Scripting (XSS)
En ese caso, los atacantes cargan *scripts* en páginas webs en la parte *frontend* para que estos se ejecuten cuando clientes reales acceden a la página. Muchos desarrolladores

olvidan añadir a las webs restricciones, como la descarga de archivos como medida de protección ante estos ataques.

Son difíciles de detectar y de identificar quién y en qué momento realizó el ataque. Y como resultado, los usuarios legítimos que visiten la web pueden descargar este código malicioso e infectar su sistema.

SQL/Command Injection (Inyección SQL)

En muchas ocasiones, se antepone el aparente correcto funcionamiento (cuando devuelve el resultado deseado) a otros criterios. Como por ejemplo, dar a los usuarios permisos elevados para que no experimenten contratiempos.

Esto abre una gran brecha de seguridad, ya que cualquier usuario tiene la capacidad de utilizar cualquier comando SQL para realizar cambios en la Base de Datos. En esta situación, un atacante puede introducir comandos SQL en la interfaz web para que sean ejecutados por el gestor de la BBDD.

Además, se suele mantener la conexión abierta con la BBDD, porque no se ha programado su cierre en ningún momento. Cualquier usuario puede encontrar estas conexiones y utilizarlas para extraer datos, por lo que la integridad de la misma se ve violada.

Use After Free

También manipula la memoria. Cuando una variable de una aplicación reserva un espacio en memoria para su uso, una vez termina, libera este espacio y se marca como memoria libre. Si el atacante conoce esta dirección, puede acceder a la lista de la memoria libre e insertar un *malware.*

La próxima vez que se asigne esta dirección -que contendrá el *malware-,* este puede ejecutarse y provocar daños. Además, permite al atacante leer el contenido de este espacio de memoria. Algunas herramientas permiten revisar la ejecución de proceso para obtener información sobre la memoria. Pero puede resultar un arma de doble filo, ya que un atacante puede utilizarla como herramienta de *hacking*.

La lista de *Mitre* contiene otros muchos ciberataques, como la autenticación inadecuada, credenciales desprotegidas o permisos incorrectos. A pesar de ello y los avances de las diferentes tecnologías, los ataques más comunes se mantienen desde el *Internet* más primigenio. Y así seguirá siendo hasta que se logre interiorizar las estrategias de un desarrollo seguro para contrarrestar estas vulnerabilidades en el *software* [37].

5.2 Mejores prácticas para SSDL

Como se ha comentado en apartados anteriores, una de las principales causas por la que se producen vulnerabilidades en el *software* es por la falta de interiorización del paradigma de seguridad durante el desarrollo. Algunas buenas prácticas que se recomiendan para que esto no suceda se explican a continuación [38].

1. Educar a los desarrolladores

Es importante mentalizar a los desarrolladores para que cambien al arquetipo del desarrollo seguro de aplicaciones. Algunas iniciativas para conseguirlo son:

- Crear unas directrices para seguir durante el desarrollo para que sean utilizadas a modo de guía durante el proyecto.
- Proporcionar al equipo la formación necesaria para realizar una codificación segura.
- Establecer de manera clara la metodología para actuar con rapidez para abordar los problemas que se van descubriendo durante la producción (*remediation SLA´s*)
- No es necesario implementar todas las acciones, cada proyecto y equipo es diferente y se deben actuar de manera dinámica buscando un equilibrio natural.

2. Definir las necesidades de forma clara

Una vez seleccionada la metodología de trabajo, debe ser fácil de entender para todo el equipo, para que sean fáciles de cumplir y actuar ante cualquier situación de la manera más recomendable.

Cualquier vulnerabilidad que sea detectada, debe contar con un método sencillo para ser solucionada. Además, todo el equipo y las herramientas utilizadas deben ser capaces de no solo detectar el problema, sino de solucionarlo.

3. Mentalidad abierta al cambio

Cuando se implantan nuevas metodologías de trabajo, siempre puede darse cierta resistencia al cambio por parte de los miembros del equipo, ya que modificará la forma de trabajar e interactuar. Personas con poca tolerancia al cambio, pueden generar dificultades durante el desarrollo del proyecto.

El equipo de seguridad debe de estar seguro de poder formar y mentalizar a los desarrolladores para generar *software* seguro gracias a sus directrices.

4. Adoptar el cambio junto a otras iniciativas

En algunas ocasiones, puede ser más sencillo realizar los cambios SLD cuando se realizan otros esfuerzos de actualización del sistema o cambio de tecnología, como puede ser una migración a la nube, la introducción de la iniciativa *DevOps* o su variante adaptada a la seguridad ***DevSecOps***.

5. Centrar primero en los problemas más importantes

Conviene centrarse en los grandes problemas antes de abordar alguna vulnerabilidad descubierta. Aunque sea una forma de actuación eficiente para nuevo *software* o aplicaciones menores, solucionar todas las fallas de seguridad que se van encontrando, es bastante más difícil de realizar en grandes o antiguas aplicaciones.

Un enfoque de triaje (división) puede ayudar en estas situaciones. No solo se centra en evitar que problemas de seguridad puedan llegar a la fase de producción si no que las vulnerabilidades detectadas se aborden y se solucionen a lo largo del tiempo.

6. DevOps

6.1 Revolución de la metodología Ágil

La metodología agile cambió por completo la forma de desarrollar *software*, donde uno de los principales objetivos es la entrega de versiones funcionales -para que los usuarios puedan testear-, adaptándose de manera repentina a los cambios de los requisitos, y para conseguirlo, la colaboración y comunicación se convierten en pilares esenciales.

Centra el foco en las entregas de *software* que se realizan a los usuarios, y cómo estas deben funcionar. Para lograr este y otros objetivos, es indispensable la colaboración entre todo el equipo y mantener una comunicación continua y fluida. De esta forma, se rompen las largas esperas de los clientes para poder tener una versión con la que poder probar el *software*, que en ocasiones, se alargaban años.

Por lo tanto, las iteraciones de las fases de trabajo, las continuas entregas de *software* funcional, junto a la comunicación con los clientes para recoger sus impresiones y adaptar las siguientes versiones a partir de ellas, forman los pilares de esta nueva metodología.

6.2 ¿Qué es DevOps?

Antes de introducir el concepto de seguridad durante el desarrollo, es importante conocer el núcleo del método, el desarrollo y las operaciones, conocido como DevOps. Comparte una estrecha relación con el desarrollo ágil, para obtener como resultado un método colaborativo entre el desarrollo y las operaciones. De esta cooperación, se obtiene un objetivo común, la creación de un *software* funcional y robusto ante cualquier vulnerabilidad para ser entregado al usuario final.

Para llevar a cabo este concepto, DevOps respalda un conjunto de procesos consolidados y bastante automatizados que promueven el *feedback* rápido y ágil, a la vez que se reducen los plazos de las entregas, instaurando un ciclo de continuas mejoras.

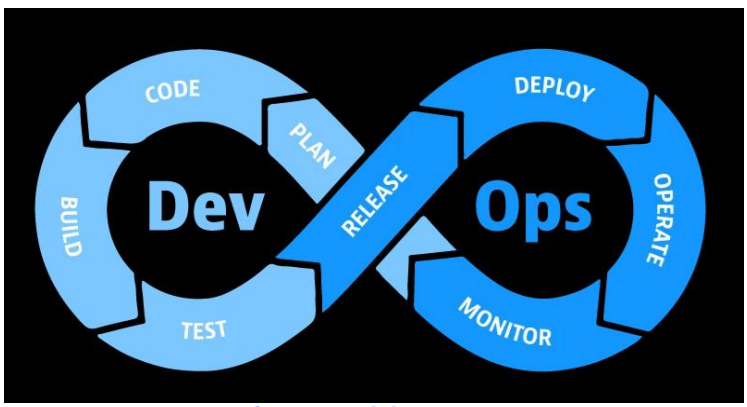


Figura 10 Ciclo DevOps

Por otro lado, para reducir los tiempos de entrega, se debe acompañar de una continua revisión para asegurar que ninguna funcionalidad o característica importante se deja de lado. Desacertadamente, la seguridad suele ser un habitual en estos casos, y aquí es donde entra en escena DevSecOps

6.3 La necesidad de DevOps

A pesar de que los equipos de desarrollo se adaptaban a la nueva metodología, la colaboración con los equipos de fases posteriores era muy baja. Esto generó una brecha entre el equipo de operaciones -el que gestiona las infraestructuras e implementa el *software* durante su lanzamiento- que desembocó en una forma muy diferente de trabajar con el grupo de desarrollo, dando como resultado una gran falta de entendimiento.

Es cierto que el equipo de desarrollo era mucho más eficiente que antes. Sin embargo, cuando se pasaba el producto a las siguientes fases, se creaba un cuello de botella. Problemas de configuración, dependencias creadas, entre otros muchos errores que se producían, terminaban en incontables consultas entre los diferentes grupos provocando tensiones entre los mismos.

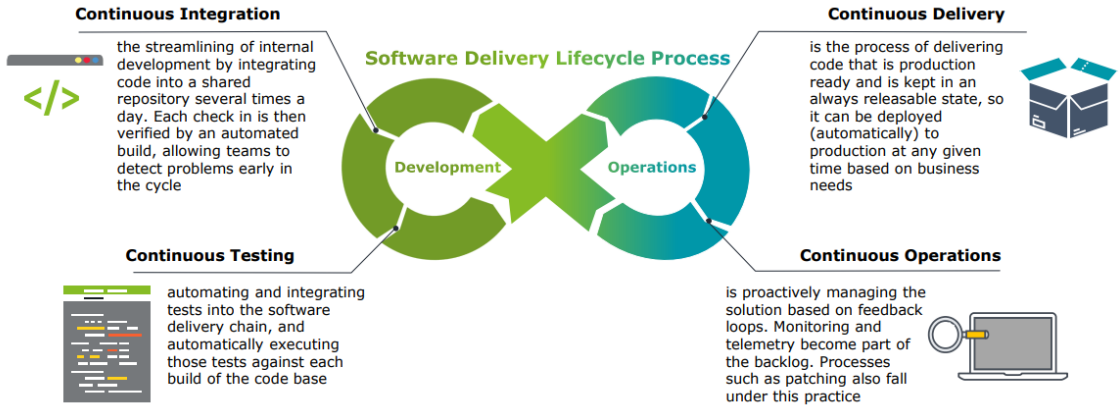


Figura 11 Proceso del ciclo de vida de entrega de software DevOps

Algunas características que definen a DevOps son:

- Un cambio cultural. La combinación entre las tareas de desarrollo y operaciones produce la integración de las actividades comunes para mejorar la eficiencia. Además, se compone de otras funciones, el resto del equipo debe trabajar de manera conjunta para entregar un *software* funcional a los clientes.

  La comunicación y la confianza entre el equipo se vuelve indispensable. Para poder realizar una entrega óptima para pasar a producción, los desarrolladores necesitan transparencia en todos los pasos. Esto lleva a extender la colaboración con otros equipos, como calidad, infraestructura o seguridad.

- Herramientas y creación de métodos automáticos. En cierta medida, los procesos manuales son eficientes a la hora de dirigir a los equipos. Sin embargo, esta eficiencia mejora significativamente cuando se introducen herramientas que son capaces de automatizar ciertas tareas, reduciendo considerablemente el tiempo en que se llevan a cabo. Automatizar ciertos métodos es indispensable en DevOps, para mejorar la calidad y no perder tiempo con elementos que serán descartados, gracias al rápido *feedback* que se genera.

- Mejora la eficiencia de los procesos. DevOps se basa en crear un desarrollo optimizado para obtener resultados gracias a los *feedbacks* más inmediatos. La responsabilidad de entregar el *software* a los clientes recae en todo el equipo, por lo que se obtiene una visibilidad total para conocer cómo se utiliza el *software* y solucionar cualquier problema planteado.

El método ágil en el que se basa DevOps, se extiende más allá de la fase de desarrollo. Las pequeñas entregas de *software* facilitan la detección de errores, junto al *feedback* continuo que se obtiene, evita derrochar esfuerzos en versiones que serán descartadas. Por lo que toda la organización se beneficia al trabajar en aplicaciones más pequeñas [39].

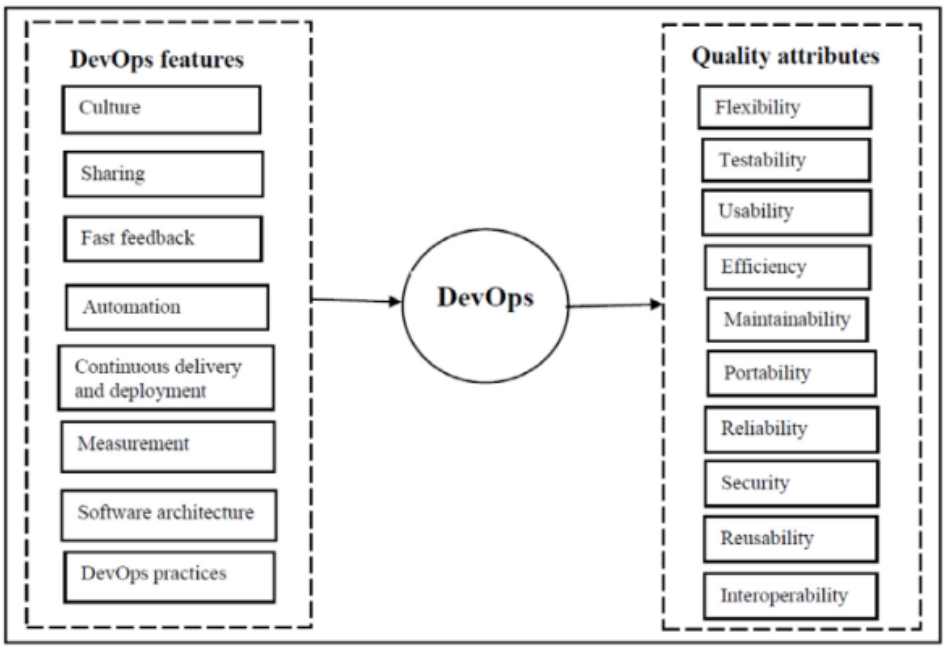


Figura 12 Relación entre las características DevOps la calidad del *software*

6.4 DevOps y seguridad

Los equipos encargados del desarrollo tienen la errónea visión de que la seguridad supone más inconvenientes y molestias que ventajas. En un principio, puede ser entendible, sus ganas y ansia de terminar su trabajo para ofrecer a los usuarios una nueva versión del *software* con las últimas funcionalidades, -más novedosa y eficaz que la última edición-, chocan de manera frontal con los informes de seguridad y posibles auditorias. Los cuales, suelen suponer revisar el código para realizar los cambios oportunos, acompañado del retraso natural que esto produce en sus entregas.

Los equipos que tienen la idea de infravalorar la implantación de la seguridad en sus aplicaciones no llegan a comprender el impacto que puede suponer para el proyecto a medio y largo plazo. Son bien conocidos los casos en los que vulnerabilidades de *software* han desencadenado la filtración de datos confidenciales (como contraseñas, números de cuentas bancarias, información delicada...). Además, desde hace tiempo se han legislado leyes relacionadas con la protección de datos de personas y entidades, las cuales, pueden conducir a reclamar responsabilidades legales junto a sanciones de nivel económico si son quebrantadas [40].

Estas vulnerabilidades, al fin y al cabo, se producen por código mal planteado o escrito. En muchos casos, estas situaciones ya se han producido con anterioridad, y por lo tanto son bien conocidas. Mientras que, en otros casos, pueden darse circunstancias inéditas que no son conocidas, que plantean nuevos desafíos. Asimismo, la ecuación se complica cuando existen dependencias de *software* de terceros, donde en la mayoría de las ocasiones no se cuenta con garantías que aseguren que no existan debilidades.

6.5 Desplazar la seguridad al inicio del proyecto

En el desarrollo en cascada (explicado en capítulos anteriores) durante su fase de prueba, se realizaban de manera manual auditorías de seguridad y calidad donde el tiempo de duración era una incógnita, ya que podía demorarse una cantidad desmesurada en relación con la vida del proyecto. Además, finalizaba con interminables informes poco atractivos.

El equipo de seguridad redactaba su propio informe destacando las vulnerabilidades encontradas y un conjunto de recomendaciones a aplicar. Era habitual tener que retocar el código para cumplir con las propuestas mencionadas en estos informes y por consiguiente, tener que pasar de nuevo por todas las etapas del modelo en cascada. Estos hechos proporcionan una idea de cómo se sentía el equipo de programadores tras aplicar las revisiones de seguridad.

El conjunto de nuevos paradigmas que aparecen con la introducción de la **metodología ágil**, junto al movimiento **DevOps**, acompañados del entorno *cloud* han cambiado la perspectiva. La velocidad por entregar una parte del *software* para que los usuarios puedan testear, se ha convertido en pilar fundamental, implementando y dando

solución a las necesidades que requiera el cliente. Aunque DevOps no tenía la intención de dejar de lado la seguridad, el ecosistema existente fomentó el distanciamiento entre los equipos de desarrollo y operaciones, dando como resultado un silo (o aislamiento) del grupo de seguridad.

Con la aparición en escena de DevSecOps, se trata de dar solución a esta situación, promoviendo la aplicación de la seguridad desde el principio del proyecto y durante todas las etapas del mismo, para generar una cultura responsable y común en todas las áreas de desarrollo que proporciona ciertas rutinas a seguir. El concepto se basa en desplazar hacia la izquierda la implantación de la seguridad, es decir, hacia las primeras fases del proyecto, ya que se adapta a las nuevas metodologías a la vez que se implantan de manera más natural las buenas prácticas [41].

6.6 Añadir la capa de seguridad a DevOps

Se puede llegar a pensar, que desplazar la seguridad al comienzo del proyecto, es básicamente proporcionar al equipo de seguridad la versión del *software* para que realicen las pruebas que vean convenientes. Pero se estaría cayendo de nuevo en limitar su implicación y en no formar el grupo unido y comunicativo mencionado anteriormente.

Esto fomenta que no se respeten e implementen las recomendaciones del equipo de seguridad, provocando tener que repetir etapas y de nuevo retrasos en las entregas. Por lo tanto, de nuevo se crea un muro entre el equipo de seguridad y el resto de participantes que impide que cualquiera de las partes alcance sus objetivos. Algunos procedimientos y herramientas pueden ser clave para la integración de la seguridad.

Nombrar a un responsable de seguridad
Compartir la responsabilidad de las tareas relacionadas con la seguridad es primordial, pero en la mayoría de las ocasiones, los miembros del equipo de desarrollo no pueden estar formándose de manera continua, ya que esto conlleva una dedicación plena. Incluir a un experto en seguridad que esté actualizado en los vectores de ataque y las últimas técnicas de desarrollo para paliarlos es vital, para que pueda trasladar estos conocimientos al resto del equipo y promover una cultura de colaboración con el equipo de seguridad.

Contar con las herramientas apropiadas
Además de los conocimientos propios, las herramientas de terceros son grandes aliados para garantizar la calidad y la seguridad del código. Cada día, estas herramientas evolucionan para proporcionar mejores servicios y trabajar de una manera más automatizada, que proporcionan los resultados a través de sencillas interfaces o directamente a un buzón de incidencias. Ofrecen el beneficio de contar con *feedback* en el menor tiempo posible.

- **Herramientas SAST** (*Static Aplication Security Testing*). Realizan el análisis sobre código estático para detectar fallas como sobrecarga del *buffer* o inyecciones SQL. Son capaces de integrarse con el IDE para ofrecer *feedback* en tiempo real, según se escribe el código, evitando futuros errores desde el comienzo. Hay que tener en cuenta que se pueden producir falsos positivos, por lo que hay que tener la capacidad de descartarlos para evitar distracciones.

- **Herramientas DAST** (*Dinamic Application Security Testing*). Tiene un enfoque de "caja negra", donde se realizan las pruebas en ejecución de la aplicación para rastrear posibles vulnerabilidades ya conocidas, como *malware*, configuraciones no seguras o inyecciones SQL. Estas herramientas suelen utilizarse en las últimas fases del proyecto, ya que se aplica en tiempo de ejecución de la aplicación.

Verificar las dependencias del *software*

Bibliotecas de código abierto y diferentes componentes pueden ser una ventana hacia vulnerabilidades. En fases iniciales, se pueden ejecutar herramientas SCA, que analiza los componentes en busca de estas vulnerabilidades. Es una tarea que se debe liberar a los desarrolladores y es importante mantener actualizada la herramienta para que sea capaz de detectar las últimas vulnerabilidades conocidas.

Equipo rojo

El nombre proviene de los videojuegos de guerra. Se basa en que algunos miembros del equipo tomen el papel de enemigo e intenten atacar a la aplicación en busca de vulnerabilidades. Para que sea eficiente, los atacantes no deben haber participado durante el desarrollo. Sin una idea preconcebida de para qué se utiliza el *software*, se puede simular de una manera más realista un supuesto ataque.

Tratar las vulnerabilidades como errores de código

El enfoque DevSecOps conlleva que todo el equipo asume la integridad de la seguridad del *software*. Por lo tanto, se debe extender la cultura de abordar las incidencias de seguridad como un error más en el código y no esperar a que se ocupe el equipo de seguridad. Gracias a ello, el equipo obtendrá la experiencia necesaria para desarrollar estas nuevas habilidades adquiridas en futuros proyectos.

No bajar nunca la guardia

A pesar de instaurar todas las buenas prácticas reconocidas durante el desarrollo, nunca se puede pensar que se está totalmente seguro contra los ataques. Realizar inversiones en herramientas como *firewalls*, monitorización o autoprotección, suelen ser buenas opciones para mantener la seguridad en todo momento.

6.7 Integración y entrega continua (CI/CD)

La demanda de los servicios digitales se ha disparado desde el 2020 y se espera que solo sea el comienzo de un prometedor periodo. Las organizaciones se encuentran en una competencia constante, donde diferenciarse de la competencia es indispensable, y la velocidad de lanzamientos para adelantarse a otros se ha convertido en habitual.

Para ello, se adoptan prácticas DevOps, como la integración y entrega continua, "CI/CD" (*Continuous Integration and Continuous Delivery*). Se trata de una serie de procesos relacionados que permite crear *software* de calidad a través de un conjunto de técnicas alineadas y automatizadas. El conjunto de estas prácticas, garantizan una colaboración más eficaz junto a una mayor eficacia de todo del ciclo de vida de desarrollo del *software*.

Agilidad en el desarrollo
La integración continua (CI) es una práctica que agiliza el proceso de creación de *software*. Permite trabajar a varios desarrolladores en diferentes módulos de la misma aplicación de manera concurrente y enviar su trabajo a un repositorio compartido al finalizar. Luego se realiza la comprobación de la compilación del código y si esta falla, se notifica. Los errores se rectifican rápidamente [46].

Ayuda a evitar el "*merge hell*" [47], que se da cuando diferentes desarrolladores realizan cambios que afecta a la compilación de la línea maestra. La fusión regular ayuda a los equipos a completar *software* más rápido y de una manera más eficiente y garantiza que siempre exista una compilación actualizada y correcta, ahorrando tiempo y recursos.

Código siempre listo para ser implementado
La entrega continua (CD) es el proceso por el que los equipos DevOps desarrollan y entregan partes de *software* a repositorios -como puede ser GitHub- o un registro de contenedores. El código siempre se mantiene en un estado de implementación y a la vez, se consiguen lanzamientos regulares y sin sorpresas para el equipo DevOps.

Los equipos DevOps pueden automatizar los *pipelines* CI/CD para mover el código entre diferentes entornos para acelerar las etapas de compilación. Por ejemplo, en el momento de realizar una demostración a un cliente, una herramienta CD puede implementar la aplicación en un servidor de prueba para que vea cómo funciona antes de su lanzamiento en el servidor de producción.

Los clientes siempre cuentan con lo último
La implementación continua (también CD) es una extensión donde las compilaciones que pasan las pruebas se implementan directamente en entornos de producción. Al ser un proceso continuo, se acelera la retroalimentación con los clientes y aligera la carga de trabajo de los equipos de operaciones.

Cuando se consigue cierta madurez en la entrega continua, se adoptan pruebas automatizadas llamadas secuencia escalonada azul-verde. Donde la nueva compilación

(verde) se implementa en paralelo con la ya existente (azul) para asegurarse que la nueva funciona antes de retirar la actual [48].

Beneficios de CI/CD

El principal beneficio que se obtiene de la integración y entrega continuas es que el tiempo de desarrollo de aplicaciones se reduce considerablemente. Esto facilita a las organizaciones una ventaja competitiva sobre sus competidoras. Además permite iterar y lanzar nuevas funciones de manera continuada. Así el que todos trabajen con la misma versión del código mientras se añaden y mejoran funciones está garantizado.

Con la implementación continua los equipos tienen que dedicar menos tiempo esperando el resultado de las pruebas y más en el desarrollo, acelerando el *feedback* de los usuarios con el objetivo de mejorar los resultados comerciales. Todas estas características hacen de CI/CD una propuesta atractiva para cualquier tipo de organización.

7. DevSecOps

DevSecOps se puede definir como la combinación óptima entre pruebas de seguridad y protección durante el ciclo de vida de la implementación y el desarrollo del *software*. Como ocurre en DevOps, está tan vinculado con la cultura de responsabilidad compartida como con cualquier tecnología específica. Además, también comparten objetivos comunes, como puede ser sacar a producción *software* de mayor calidad en un tiempo mucho menor al habitual o detectar y subsanar las fallas que van apareciendo de una manera rápida y eficaz [49].

En un principio puede sonar sencillo de llevar a cabo, pero introducir una cultura de trabajo para satisfacer estas técnicas, puede llegar a ser muy complejo e incluso imposible en algunos casos. En los siguientes apartados se analizarán los procesos y procedimientos para mejorar la comprensión de esta forma de trabajo del desarrollo de *software* seguro.

7.1 ¿Qué es DevSecOps?

DevSecOps es una táctica perfecta que conecta tres disciplinas diferentes: desarrollo (*Developer*), seguridad (*Security*) y operaciones (*Operations*). Con el objetivo de **incorporar la seguridad** de manera natural durante el proceso de integración y entrega continua (CI\CD), ya sea en entornos de preproducción (desarrollo) o producción (operaciones). A continuación, se describe el papel que toma cada una de las disciplinas.

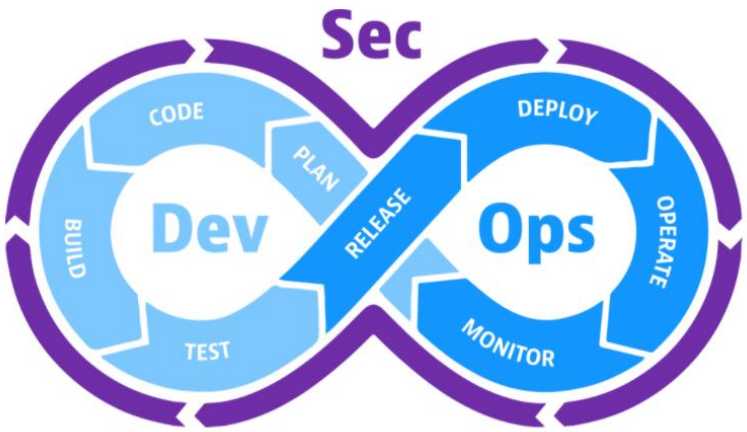


Figura 13 Ciclo DevSecOps

Desarrollo
El equipo de desarrollo genera y realiza continuas mejoras durante las iteraciones de las nuevas aplicaciones, cabe destacar:

- Aplicaciones personalizadas, adaptadas y diseñadas con un propósito específico que satisfagan las necesidades de los clientes.
- Aplicaciones que obtienen todos los beneficios que pueden ofrecer las fuentes de código abierto con el objetivo de acelerar el proceso de desarrollo.
- Conexiones promovidas por API´s que reducen las brechas generadas entre los nuevos servicios y los sistemas heredados.

Las técnicas de desarrollo actuales se basan en modelos ágiles, donde se otorga prioridad a una mejora continua ante los pasos secuenciales e inamovibles del desarrollo en cascada. Si la metodología de trabajo por parte de los desarrolladores se produce de forma aislada –sin tener en consideración las operaciones y la seguridad-, el nuevo *software* puede presentar problemas de operatividad o vulnerabilidades de seguridad que pueden dar como resultado un elevado coste (tanto en tiempo como es esfuerzo) para ser solventadas.

Operaciones
Esta especialidad se encarga de realizar los procesos que gestionan las diferentes funcionalidades que realiza un *software* durante el tiempo que se extiende su ciclo de vida, tanto de entrega como de uso. Se pueden incluir:

- Monitoreo que ejerce el sistema.
- Solución de las taras o deficiencias.
- Realizar las pruebas necesarias para asegurar el comportamiento deseado tras aplicar cambios o actualizaciones.
- Ajuste del sistema durante el lanzamiento del *software*.

Durante los últimos años, la metodología DevOps se ha ido imponiendo a la hora de combinar ciclos de desarrollo con principios operativos clave. En algunas ocasiones durante operaciones posteriores al desarrollo, se puede requerir su realización en silos para facilitar la identificación de ciertos problemas, junto a su solución.

Sin embargo, este planteamiento estanca el avance de nuevos desarrollos, ya que obliga a solucionar los problemas de *software* primero. Este hecho complica la situación en lugar de facilitar un flujo de trabajo optimizado. Implementar las operaciones con los procesos de desarrollo de manera paralela otorga una reducción en los tiempos y un aumento de eficiencia en muchos aspectos.

Seguridad
La seguridad engloba el conjunto de herramientas y técnicas utilizadas a la hora de diseñar y elaborar *software* resistente a ataques y capaz de sobreponerse ante intrusiones reales en el menor tiempo posible.

A lo largo del tiempo, lo más habitual ha sido tratar la seguridad de las aplicaciones una vez ha finalizado su desarrollo completo. Además, los encargados de esta tarea, eran un conjunto de personas aisladas del resto de los equipos (desarrollo y operaciones) que habían participado en la creación de la aplicación. Este planteamiento de silos provocaba desarrollos más largos y un aumento en los tiempos de reacción a la hora de solventar problemas.

También cabe destacar que las herramientas utilizadas para la seguridad siempre han estado aisladas. Cada prueba se centraba en una aplicación específica incluyendo su código fuente, anulando la generalización de procesos. La visión de los problemas de seguridad que iban surgiendo se limitaba al conocimiento del equipo de seguridad, por lo que el resto de miembros nunca llegaba a entender los riesgos que corría el *software* en un entorno de producción.

Cuando la seguridad de una aplicación se integra en el proceso DevSecOps, -desde los inicios hasta su salida a producción-, las organizaciones tienen la capacidad de implementar de manera conjunta los tres componentes más importantes de la creación y entrega de *software*. De igual manera, permite realizar las pruebas de seguridad de una manera transparente, automática y en paralelo a otros desarrollos o pruebas.

Por ejemplo, se pueden ejecutar pruebas de seguridad durante el propio desarrollo de la aplicación, sin producir contratiempos al trabajo de otros equipos. De la misma manera, se pueden ejecutar pruebas durante la fase de producción, con la intención de encontrar vulnerabilidades para corregirlas poco después de ser localizadas.

7.2 Desafíos en la implementación de DevSecOps

Como era de esperar, durante la implantación de DevSecOps se pueden presentar diferentes desafíos. Algunos de los más habituales son los siguientes.

Equipo, cultura y resistencia al cambio. A pesar de que los miembros del equipo desempeñan un gran trabajo como DevOps, es habitual que tengan que volver a tomar formación para que adquieran los conocimientos y comprendan ciertas técnicas de seguridad, para operar de manera eficaz con las nuevas herramientas de seguridad con las que trabajarán.

Deben asimilar y adoptar el papel de que son los responsables de implementar la seguridad en el *software* que desarrollan, a la vez que mantengan su funcionalidad y usabilidad. Este hecho requiere un **cambio cultural** que podría encontrar resistencia. El equipo está acostumbrado a priorizar la velocidad de lanzamiento frente a presentar una aplicación que cumpla con los niveles de seguridad estipulados.

Se debe animar a todos los miembros del equipo a colaborar, dando la misma importancia a la seguridad como a la velocidad de lanzamiento de un nuevo ciclo de desarrollo. Cuando asimilan la importancia que tiene su participación, serán capaces de incorporar ideas innovadoras, convenientes para afrontar los problemas de seguridad sin verse perjudicada la velocidad del desarrollo.

Definición de políticas de seguridad para los desarrolladores. Los *pipelines* de DevSecOps suelen tener un equipo dedicado a la seguridad que define las políticas de la organización [56]. Las cuales, pueden determinar reglas de cifrado, normas de uso de SAST, DST o SCA o mejores prácticas de codificación.

Cuando el equipo de desarrolladores tiene claro el conjunto de pautas que deben seguir, conocen en todo momento lo que deben, o no deben hacer, a la vez de como adaptar el trabajo para que la seguridad de las aplicaciones mejore día a día.

Herramientas adecuadas. Hay que identificar qué herramientas son las más apropiadas para aplicar en cada fase y proyecto, con la finalidad de que su integración en el flujo de trabajo sea lo más natural y eficiente posible. Contra más automatizadas están las herramientas elegidas y más integradas con el CI/CD, menos formación y cambio cultural requerirá el equipo.

Sin embargo, herramientas demasiado automatizadas pueden no ser siempre la mejor opción. Es habitual que los entornos de desarrollo experimenten grandes cambios a lo largo de los años, junto a que muchas de las herramientas actuales están compuestas en su mayor parte por código abierto. Lamentablemente, en innumerables ocasiones el *software* de código abierto no es capaz de realizar detecciones precisas ante posibles vulnerabilidades.

En arquitecturas basadas en la nube, sucede algo similar. Las aplicaciones nativas *cloud* se ejecutan en contenedores, donde las herramientas de seguridad tradicionales para trabajar en entornos *on premise* -incluso herramientas clasificadas para "seguridad *cloud*"- no están capacitadas para realizar evaluaciones precisas de las aplicaciones de estos contenedores.

7.3 Principales características de las prácticas exitosas de DevSecOps

Los objetivos principales de DevSecOps definidos anteriormente, son publicar *software* de mayor calidad en un tiempo mucho menor y detectar y sobreponerse a posibles fallas durante el periodo de producción de manera rápida y eficiente. Para alcanzar

estos objetivos de manera solvente, se deben cumplir algunas características para establecer un programa DevSecOps robusto.

1. **Sensibilización responsabilidad de la seguridad**. Todos los miembros de los diferentes equipos que participen en el desarrollo deben conocer los fundamentos de seguridad y sentirse responsable de los resultados obtenidos. El ideal que debe calar hondo es el de "**la seguridad es responsabilidad de todos**", es uno de los pilares que forman las bases de la cultura DevSecOps.

2. **Operaciones automatizadas**. La mayoría del conjunto de herramientas de CI/CD cuentan con un alto grado de automatización. Las herramientas DevSecOps no pueden quedarse atrás en este aspecto. Estas deben contar con una automatización completa, donde los pasos manuales deben ser nulos o prácticamente ninguno. En todo momento se debe facilitar la información relevante sobre la seguridad de la aplicación, incluso cuando los desarrolladores quieran evitar pruebas que puedan ralentizar su trabajo.

3. **Integrar con canalizaciones de CI/CD.** El objetivo es descubrir problemas de calidad y generar un ciclo rápido de retroalimentación a los desarrolladores. Este principio se aplica a la seguridad.

    - CI/CD debe ejecutar herramientas de seguridad de manera continua y automática.
    - Se registran los cambios del código y aplicaciones para obtener métricas e informes.
    - Los cambios que generen problemas de seguridad lanzan respuestas automáticas.
    - Las herramientas de colaboración utilizadas por el equipo de seguridad deben ser compartidas con los equipos de desarrollo y operaciones.

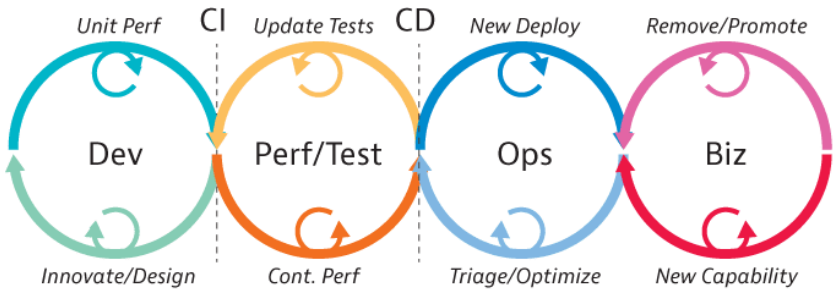


Figura 14 Ciclo de vida SDLC automatizado del *software*

4. **Resultados en tiempos reducidos**. Es conveniente que las herramientas generen resultados prácticamente en tiempo real, ya que la velocidad es otra de las piedras angulares de un equipo DevOps actual.

5. **Alcance extenso y poder de adaptación**. Las herramientas de seguridad deben ser capaces de funcionar y no limitar sus capacidades en diferentes entornos,

como contenedores *cloud*, PaaS, Kubernetes, nubes híbridas... Sin generar silos o puntos ciegos.

A la vez, las herramientas están obligadas a proporcionar información sobre cualquier tipo de aplicación, incluidas las que se basan en código abierto, así como aplicaciones adquiridas de terceros, de las que no se posee su código fuente.

6. **Shift-left y shift-right**. Ya se ha comentado en varias ocasiones los beneficios que aporta realizar las evaluaciones de seguridad desde el principio del proyecto, para intentar evitar que posibles vulnerabilidades lleguen a la etapa de producción. Pero también hay que tener en cuenta las últimas etapas de desarrollo (producción) o "desplazamiento a la derecha".

    - En producción es donde se generan la mayoría de los ataques.
    - El código fuente otorga mucha información. Sin embargo, la que se obtiene mientras la aplicación se ejecuta en el entorno de producción, "vale su peso en oro".
    - En algunas ocasiones, aplicaciones ejecutadas en el entorno de producción no se ejecutaron en el de desarrollo, y por tanto, no llegaron a ser analizadas por las herramientas de seguridad específicas.
    - Para detectar posibles vulnerabilidades en tiempo real, las aplicaciones deben ser monitorizadas en el entorno de producción.

7. **Precisión**. La automatización es clave, pero debe estar acompañada de calidad y precisión. En estudios recientes [57], se comprobó que la mayoría de las alertas de seguridad que detectan las herramientas resultan ser falsos positivos, los cuales no requieren ninguna acción. Para que DevSecOps resulte eficaz, hay que recurrir a pruebas que sean capaces de detectar cuándo ocurren los falsos positivos y los falsos negativos, ofreciendo información valiosa al equipo.

8. **Aceptación del desarrollador**. Todos los elementos que forman el programa DevSecOps tiene que ser aceptado por los miembros del equipo de desarrollo, ya que tienen que ser capaces de realizar las pruebas necesarias, encontrar vulnerabilidades y poner solución a cualquier problema.

9. **Trazabilidad, auditabilidad y visibilidad**. Clave para el éxito de DevSecOps.

    - **Trazabilidad**. Seguimiento de los cambios de configuración y entorno desde la planificación hasta la producción.
    - **Auditabilidad**. Generar de manera automática los informes de los procesos de desarrollo y los controles de seguridad.
    - **Visibilidad**. Comprensión de lo que se ejecuta en el entorno, identificar las amenazas y responder ante ellas.

7.4 Formas de integrar la seguridad en el ciclo de desarrollo

DevOps contribuyó a la automatización de procesos optimizados como método de trabajo con el fin de mejorar la calidad del *software* a la vez que reduce el tiempo en que este sale a producción. DevSecOps agrega la capa de seguridad al conjunto y elimina los posibles silos entre los tres equipos que participan en el desarrollo. Para garantizar que este entorno cuente con las mejores prácticas y pruebas de seguridad, desde su planificación y desarrollo hasta su puesta en escena.

Por esta automatización, las herramientas se convierten en un elemento fundamental para DevSecOps, ya que en un entorno DevOps, la seguridad debe automatizarse e integrarse de manera estrecha con CI/CD.

Las herramientas tienen dos objetivos principales.

- Minimizar el riesgo de canalizaciones de desarrollo pero sin llegar a reducir la velocidad del proceso. Este objetivo se consigue realizando pruebas de seguridad de manera continuada para detectar y solventar las posibles vulnerabilidades de seguridad.
- Apoyo total al equipo de seguridad. Se debe permitir la supervisión de los proyectos de desarrollo sin necesidad de revisión y aprobación por parte del equipo de seguridad de manera manual.

Diferentes formas de integrar seguridad en los ciclos de desarrollo
DevSecOps es un paradigma joven y en algunos aspectos no está todavía formalizado, como puede ser un conjunto de herramientas oficiales. A continuación, se presentan algunas herramientas que las organizaciones suelen utilizar de manera habitual para integrar la seguridad en los procesos de desarrollo.

Escáner de vulnerabilidades de código abierto
Es muy habitual encontrar componentes con dependencias externas en la gran parte de proyectos de desarrollo de *software*. Estos elementos se componen de código abierto, por lo que pueden estar expuestos a vulnerabilidades de seguridad o encontrar problemas de licenciamiento una vez se incorporen al proyecto.

Este tipo de escaneos (también conocidos como análisis de composición de *software* (SCA)), analiza los elementos de código abierto, sus bibliotecas o sus dependencias existentes en el código. Cualquiera de estos elementos categorizado como código abierto, se registra tomando datos como su versión, fuente, plataforma y otras características que puedan ser de interés.

Una vez almacenados los registros, se pueden comparar con las BBDD de vulnerabilidades conocidas, las publicaciones de proveedores de *software* u otros recursos de seguridad disponibles en ese momento. Gracias a estos elementos, se pueden evaluar la gravedad de las vulnerabilidades, clasificar su impacto potencial y recomendar las posibles soluciones para reparar la situación.

El análisis de riesgos potenciales se puede aplicar durante las primeras etapas de planificación para identificar los componentes más seguros o los que están libres de vulnerabilidades. Posteriormente, los escaneos en busca de debilidades se aplican en las diferentes etapas del desarrollo y compilación. De esta manera, se puede garantizar que nuevas vulnerabilidades no se filtren tras la etapa de planificación.

Algunos beneficios de estos escaneos son:

- Escaneo de desarrollo. Los desarrolladores reciben las notificaciones cuando se detectan problemas de seguridad ocasionados en los componentes que se han visto comprometidos. A continuación, pueden tomar decisiones en un tiempo menor y más analizadas para abordar o evitar estos riesgos.

- Análisis de pruebas de seguridad. Previamente, se establece un umbral de riesgo que al ser superado por las vulnerabilidades que sufre algún componente, se generan alertas para su conocimiento y posterior análisis. Estas alertas pueden lanzar acciones automáticas de corrección o ser analizadas personalmente por los miembros del equipo de seguridad.

- Escaneo de pre y producción. Cualquier nuevo riesgo detectado en la aplicación después de la revisión de seguridad, será alertado. Estos incluyen a los elementos que se añadieron al proyecto por otros medios diferentes a SDLC o CI/CD.

Pruebas de seguridad de aplicaciones estáticas (SAST)
Estas pruebas permiten a los desarrolladores escanear el código en busca de debilidades y problemas de seguridad que deben abordarse. Cada incidencia tiene un nivel de gravedad para que los desarrolladores puedan clasificarlas y dar prioridad a las más críticas.

Cuando SAST se integra con SDLC o en un CI/CD los equipos de seguridad pueden definir niveles de calidad para identificar el tipo y el grado de los problemas que haría fallar a la aplicación. También se consigue que un componente comprometido no se extienda a las siguientes etapas CI/CD. La integración en el entorno de desarrollo (IDE) permite a los desarrolladores ver las debilidades del código en tiempo real mientras se está escribiendo, fomentando la seguridad desde los inicios.

Pruebas de seguridad de aplicaciones dinámicas (DAST)
Son capaces de realizar de manera automática pruebas de seguridad en aplicaciones en tiempo de ejecución. Son capaces de probar una gran variedad de amenazas sin necesidad de acceder al código fuente. Son muy utilizadas en interfaces HTTP y HTML en aplicaciones web.

DAST se puede definir como un método de caja negra, ya que identifica las vulnerabilidades desde el lado del atacante, recreando como un atacante lo haría de

forma real. La automatización de DAST y su fácil integración con otras herramientas la convierte en un gran aliado para verificar la seguridad en entornos de prueba.

Pruebas de seguridad de aplicaciones interactivas (IAST)
Se trata de un enfoque híbrido que combina SAST y DAST. Son capaces de escanear las aplicaciones en etapas de desarrollo (enfoque caja blanca) y en tiempo de ejecución (enfoque caja negra). Aporta ventajas como identificar vulnerabilidades en fases tempranas como en fases de desarrollo. Algunas funcionalidades son simular ataques complejos, realizar análisis dinámicos recursivos o reducir los falsos positivos.

Análisis de composición de software (SCA)
Las herramientas SCA analizan el código fuente y crean una lista de los componentes, centrándose en los de código abierto. Después, genera un árbol con sus dependencias en busca de vulnerabilidades.

Estas herramientas pueden identificar vulnerabilidades conocidas de código abierto, identificar componentes que necesitan parches o no están actualizados o realizar escaneos del código binario.

Autoprotección de aplicaciones en tiempo de ejecución (RASP)
Son consideradas la evolución de SAST, DAST e IAST. Analiza el tráfico de las aplicaciones y el comportamiento de los usuarios en tiempo de ejecución para prevenir amenazas a través del análisis del código fuente. Defiende al *software* de manera **proactiva** emitiendo alertas o cerrando la sesión del usuario. La contra es que no es capaz de sustituir un proceso completo DevSecOps ni la detección de vulnerabilidades en las primeras fases.

Orquestación de pruebas de seguridad de aplicaciones (ASTO)
ASTO es una nueva categoría de aplicaciones introducidas por ***Gartner*** en su cuadrante mágico [58]. Apoya los procesos DevSecOps durante todo el ciclo de vida del proyecto. No solo coordina y automatiza diferentes herramientas, además centraliza los datos y el conocimiento adquirido de forma que tenga fácil acceso. Los seguimientos y de los posibles riesgos son muy sencillos de analizar al concentrarlos en una única consola.

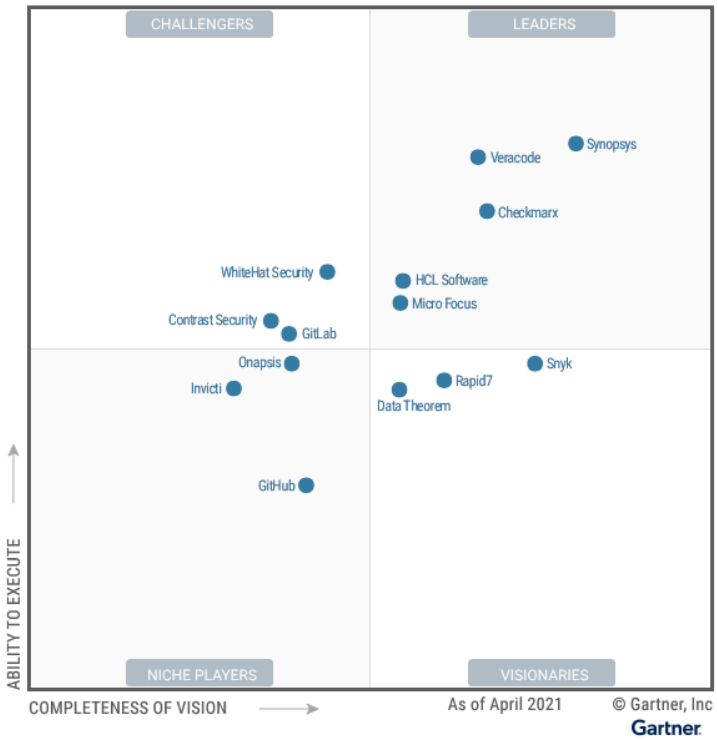


Figura 15 *Magic Quadrant for Application Security Testing*

Escaneo de imágenes

Es habitual que los equipos DevOps implementen artefactos a través de imágenes y contenedores *cloud*. Una de las principales preocupaciones en DevSecOps es identificar las vulnerabilidades en estas imágenes, ya que es habitual que se obtengan desde repositorios públicos o fuentes poco confiables.

Las imágenes pueden contener elementos *software* desactualizado o con vulnerabilidades. Para asegurar que estas imágenes son seguras, los escáneres verifican los códigos fuente y su cumplimiento en las mejores prácticas para configuraciones seguras.

Gestión de identidad y acceso (IAM). Está formado por métodos que utilizan políticas centralizadas que controlan el acceso a los datos, aplicaciones y otros elementos en red. Gracias al control realizado en cada etapa del SDLC, se previene el acceso no autorizado. Algunos métodos son:

- Control de autenticación. Verifica la identidad del usuario o servicio.
- Control de autorización. Otorga el acceso a recursos a usuarios autorizados.
- Control de acceso basado en roles (RBAC). Otorga el mismo acceso o permiso colectivo a los usuarios que tengan asignado el mismo rol.
- Módulo de seguridad de *hardware* (HSM). Dispositivos físicos que ayudan a proteger secretos (certificados, credenciales o claves)
- Firma de imágenes de contenedores. Validan la autenticidad y seguridad de las imágenes de contenedores.

Segmentación y controles de red. Permiten controlar el tráfico administrado por las herramientas de orquestación de contenedores. Asegura el flujo de comunicación entre los elementos de las aplicaciones en contenedores con microservicios. Además, al segregar las redes con diferentes finalidades (pruebas, producción...) se garantiza que un ataque a uno de los entornos no afecte a los demás.

Control de los datos. Se busca proteger la integridad de los datos y evitar posibles filtraciones. Algunos métodos utilizados son el cifrado de datos (clave privada y simétrica, *tokens* y gestión de claves); protección de datos (clasificación de los datos que proporcionan controles de seguridad para mejorar el cumplimiento normativo) y enmascarar datos (la anonimización de datos proporcionan datos realistas en entornos de desarrollo y pruebas)

Herramientas de automatización de infraestructura
Los enfoques modernos se basan en automatizar la seguridad y las configuraciones de la infraestructura. Estas herramientas son capaces de detectar y reparar de manera automática las vulnerabilidades de seguridad y los problemas de configuración en entornos *cloud*. Se mueven desde la automatización basada en eventos, hasta la gestión de la infraestructura como código (IaC) o plataformas de protección de carga de trabajo en la nube (CWPP).

Herramientas de visualización y panel
Los equipos DevSecOps requieren de herramientas que permitan centralizar la información relevante de seguridad para poder ser compartida con los demás integrantes del equipo. También se puede integrar en otras herramientas de gestión.

Este tipo de herramientas se ha comprobado que resultan muy interesantes para el equipo, ya que de una manera muy sencilla, se visualiza el crecimiento o la reducción de las vulnerabilidades a lo largo del proyecto. A los paneles de visualización, se pueden agregar datos de seguridad, registro y una gran variedad de estadísticas visibles para todos los miembros participantes.

Herramientas de modelo de amenazas
Estas herramientas permiten a los equipos DevSecOps predecir, detectar y evaluar amenazas de toda el área de ataque. El objetivo marcado es tomar decisiones lo más rápido posible de manera proactiva basadas en la información obtenida para reducir los riesgos al mínimo. Hay un gran número de estas aplicaciones en el mercado con grandes capacidades.

Herramientas de alerta
Estas herramientas ayudan a los equipos DevSecOps a dar una respuesta inmediata ante los eventos de seguridad. En un escenario ideal, se notifica cada alerta sólo cuando el evento ha sido analizado y considerado meritorio de informar al equipo. Es básico para no sobrecargar el sistema y generar interrupciones continuas en los flujos de trabajo. Una vez de lanza la notificación, el equipo analiza el evento y aplica las correcciones necesarias.

7.5 Herramientas DevSecOps

Github Actions
Es una herramienta de código abierto que trata de automatizar los *worksflow* de DevSecOps. Permite a los miembros del equipo escribir, probar y desplegar código desde GitHub [59]. A la vez que en tiempo real realiza acciones propias de GitHub como solicitudes de extracción o creación de nuevas versiones. Es utilizada por el equipo de desarrollo y proporciona la automatización de diferentes tareas [60].

También proporciona un entorno de ejecución nativo gratuito, el cual puede ofrecer más opciones mediante el pago de tarifas. Las acciones con las que cuenta incluyen funcionalidades de flujos de trabajo, gestión de registros o ejecuciones de los principales Sistemas Operativos. Tanto los flujos como las acciones se pueden definir con la sintaxis YAML [61]. Aporta agilidad en el desarrollo dentro de la integración y entrega continua (CI/CD)

Trivy
Es un escáner de vulnerabilidad de código abierto orientado para imágenes de contenedores en la nube. Sus puntos fuertes son su facilidad de despliegue y que no tiene la necesidad de descargar una BBDD de vulnerabilidades. Trivy es capaz de detectar vulnerabilidades de Sistemas Operativos y dependencias sobre aplicaciones existentes en imágenes de contenedores. Es adecuado para los *pipelines* DevSecOps que se integran con herramientas CI (CircleCI o GitLab) [62].

El equipo de desarrollo puede hacer uso de la herramienta para automatizar tareas de detección de vulnerabilidades en el menor tiempo posible.

StarBoard
Kubernetes se utiliza de manera habitual en dominios DevSecOps para orquestar *clusters* de contenedores. StarBoard integra herramientas de seguridad sobre en el entorno Kubernetes, para encontrar los riesgos asociados de los recursos de una manera nativa. Ayuda al equipo en la labor de la entrega continua (CD).

StarBoard proporciona recursos personalizados y un módulo Go para utilizar en los escáneres de seguridad, junto a comandos compatibles con Kubectl, que permite acceder a los informes generados de seguridad de manera nativa desde kubernetes [63].

OWASP Zed Attack Proxy (ZAP)
Es una herramienta popular que ayuda a los desarrolladores a desplegar una seguridad de mayor calidad en el *software*. La herramienta cuenta con diferentes características, como un escáner activo que se integra en el pipeline CI/CD. Para mejorar la seguridad, ZAP utiliza un servidor proxy para enrutar el tráfico de un sitio web. Es capaz de detectar vulnerabilidades de los sitios web [64].

HashiCorp Vault
Esta herramienta permite el acceso seguro a secretos -como claves API, contraseñas o certificados-. Registra los accesos en forma de auditoría y proporciona una interfaz para todos los secretos existentes en la infraestructura. Colabora en el cifrado de los datos para asegurar la integridad de la información (uno de los pilares de la seguridad).

También puede generar de manera dinámica secretos para servicios o sistemas específicos como BBDD o servicios *cloud*. De la misma manera se pueden revocar cuando el servicio ya no tiene acceso a los mismos. Otra característica, es que es capaz de cifrar secretos sin necesidad de almacenarlos, lo que permite al equipo de seguridad almacenar información sensible en el sistema sin tener que gestionar el cifrado [65].

SonarQube
Es una herramienta de código abierto que es capaz de realizar de manera desatendida revisiones en el código. Su objetivo es detectar vulnerabilidades y errores en el código fuente. Es compatible en un gran número de lenguajes de programación y puede integrar los pipelines de DecSecOps para garantizar el acceso a todos los colaboradores a la retroalimentación que genera la herramienta de manera continua [66].

Aqua Security
Proporciona herramientas DevSecOps que ayudan a automatizar el desarrollo y despliegue de aplicaciones nativas en la nube sin aumentar la carga de recursos existentes. Integra una serie de herramientas para gestionar las vulnerabilidades, el escaneo de la configuración de la seguridad en entornos *cloud*, en entornos Kubernetes, detección de *malware* preproducción y gestión de políticas para ratificar la seguridad extremo a extremo de DevSevOps [67].

Snyk
Es una herramienta Open Source que permite realizar una continua auditoría a los desarrollos de *software*, escanea las librerías en tiempo real y genera los reportes con los resultados. Muchos desarrollos cuentan con componentes de código abierto, donde lamentablemente las vulnerabilidades de las bibliotecas utilizadas no paran de aumentar [68]. Snyk se especializa en evaluar el código de varios lenguajes con una búsqueda automatizada y reparar las vulnerabilidades que presentan las dependencias. Es capaz de realizar estas tareas directamente desde plataformas de control de versiones como Github [69].

SonarCloud
Se trata de una plataforma de análisis continuado del código realizado de manera *online* de los proyectos de desarrollo de *software*, y que ofrece la revisión de los resultados desde la nube. El objetivo es prevenir y responder de manera eficaz ante las amenazas de seguridad en contra de las aplicaciones *software*. Eso incluye todas las etapas del proyecto (desarrollo, diseño…) como el conjunto de sistemas y metodologías utilizadas por las aplicaciones tras su despliegue. Las herramientas que ofrece, fomentan a generar código llamado "*Continuous Code Quality*". También ayuda

a mejorar la entendibilidad del código, gracias a su métrica de complejidad cognitiva [70].

7.6 Seguridad de las aplicaciones

Con el desplazamiento a la izquierda de la seguridad desde la llegada de DevSecOps, se ha generado un enfoque adicional para garantizar la seguridad de las aplicaciones desde las primeras etapas del desarrollo.

Los diferentes equipos participantes colaboran para identificar los problemas de seguridad durante todo el ciclo de vida del desarrollo, con la intención de introducir de manera natural, su solución durante los flujos habituales de trabajo. Cuanto antes se solucionen, más se reduce el riesgo de explotar el *software* por parte de un atacante.

Ya es habitual recibir alertas de las vulnerabilidades mientras se escribe el código, para ser corregidas antes de pasar a las siguientes etapas con las herramientas CI/CD. La seguridad del *software* se integra durante todas las etapas del proceso, incluida la fase de producción donde implementan en tiempo de ejecución.

Algunas buenas prácticas aconsejables para la seguridad de aplicaciones son:

Evaluación de amenazas
Se deben clasificar las aplicaciones por rango de sensibilidad a ataques e investigar sus puntos débiles, los que podría explotar el atacante contra el *software*. Hay que analizar las medidas de seguridad implantadas en la actualidad y determinar si son las más adecuadas. Establecer objetivos realistas y adecuados, es clave para mejorar la protección y alcanzar el nivel de seguridad que requiere la aplicación.

Integrar la seguridad en los procesos CI/CD
Las metodologías de desarrollo de *software* de la actualidad, se gestionan a través de herramientas de integración continua/entrega continua (CI/CD) capaces de automatizar el proceso de lanzamiento. Todas las herramientas de seguridad tienen que estar integradas en los procesos CI/CD.

Estas herramientas de automatización, permiten que las aplicaciones puedan ser probadas en diferentes puntos de control. Por ejemplo, cuando se compila el código fuente, este se somete de manera automática a las pruebas de seguridad configuradas, enviando un reporte a los desarrolladores para que éstos corrijan los problemas de seguridad que contiene el código.

Dar prioridad a las correcciones
En la mayoría de los proyectos, las herramientas de seguridad detectan un gran número de vulnerabilidades, donde no siempre es posible dar solución a todas ellas. Los equipos deben tener la capacidad de identificar las vulnerabilidades más críticas, para ello, los procedimientos deben ser eficientes para no poner en riesgo la productividad de los desarrolladores.

El proceso de pruebas de seguridad debe incluir indicadores automatizados, sobre la gravedad y el potencial de explotación de cada vulnerabilidad. Si es necesario, se puede realizar una evaluación manual, para entender si las vulnerabilidades son realmente un riesgo para el negocio. Por ejemplo, un componente vulnerable puede no utilizarse en absoluto en la aplicación de producción, o un sistema vulnerable puede tener otras medidas de seguridad que dificulten su explotación.

Gestionar los privilegios de manera adecuada

Las herramientas de seguridad -como políticas, procesos o credenciales- suelen ser sensibles para ser aprovechadas por atacantes. Varios ataques a la cadena de suministros, que resultaron desastrosos (como el caso de SolarWinds [71]) fueron posibles por debilidades en los procesos de CI/CD.

Por defecto se deben otorgar los permisos mínimos para que los usuarios tengan acceso sobre los datos imprescindibles y necesarios para que sean capaces de realizar su trabajo. Es muy buena práctica configurar MFA (*Multi Factor Access*) y supervisar los accesos en busca de comportamientos sospechosos.

Pruebas de penetración

Las pruebas autorizadas son de gran ayuda para solucionar ciertos problemas. Sin embargo, algunas vulnerabilidades pueden pasarse por alto. Es práctico utilizar probadores de penetración humanos, es decir, *hackers* éticos que evalúan las aplicaciones desde el papel de un atacante. Detectan vulnerabilidades e intentan romper la aplicación sin causar daños reales.

También cabe la posibilidad de realizar penetraciones a ciegas, que son aquellas que se producen sin avisar los equipos de seguridad y operaciones. Se pueden comprobar los procedimientos y las habilidades del personal de seguridad para contrastar resultados en caso de que hubiera sido un ataque real.

Comprobar componentes de terceros

El *software* actual se compone por un gran número de componentes de código de terceros, de los cuales muchos son de código abierto. El código abierto tiene muchas ventajas y a la vez corre el peligro de exponer a las compañías en riesgos de seguridad.

Con los proyectos de código abierto que no reciban el mantenimiento necesario y sus prácticas de codificación no sean las más seguras, se debe analizar la situación y tomar alguna medida para paliar el problema. Para proporcionar una confianza básica, deben actualizarse de manera regular con el fin de evitar vulnerabilidades.

Una buena solución para escanear componentes de terceros es *Software Composition Analysis* (SCA). Las aplicaciones SCA escanean los componentes de código abierto en busca de vulnerabilidades y problemas de licencias que puedan amenazar al desarrollo del proyecto. Ofrecen soluciones detalladas para ayudar a resolver los problemas o reemplazar los componentes comprometidos de código abierto [72].

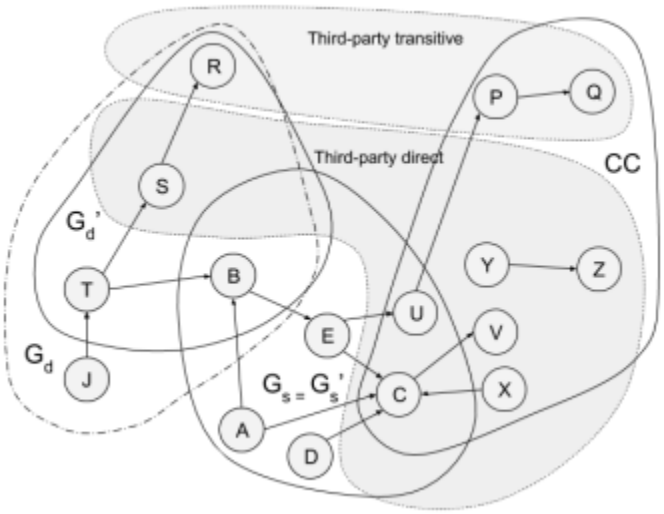


Figura 16 Ejemplo de método de dependencias de *software* de terceros

7.7 Seguridad de la cadena de suministro

La seguridad de la cadena de suministro ofrece a las organizaciones la posibilidad de detectar y mitigar los riesgos de los artefactos digitales que ingresan en el *software* a través de terceros. Estos pueden ser bibliotecas de *software* o desarrollo subcontratado. Una estrategia integral que combina la gestión de riesgos junto a los principios de seguridad, implementa las medidas necesarias para bloquear o mitigar estos riesgos.

Un ataque a la cadena de suministro puede ser un intento de infiltrarse en el *software* o en entornos *cloud*. Por ejemplo, el atacante puede inyectar *malware* camuflado en una actualización o contribuir con una parte de código malicioso a un proyecto de código abierto. Al confiar en este *software*, los usuarios lo incorporan a sus proyectos y *pipelines* CI/CD e implementan el *malware* de manera inconsciente.

Vulnerabilidades de cadenas de suministro físico y de *software*

Cadenas de suministro físico. Esta cadena incluye todos los elementos y procesos físicos para crear el producto físico final, que es entregado tras la compra del mismo. Pueden ser los procesos como fabricación o logística.

Cadenas de suministro de *software*. Los proyectos de *software* se componen de elementos listos para usar, ya sea de código de terceros, abierto, personalizado o proporcionado por APIs.

Su ventaja es que puede acelerar el desarrollo de las aplicaciones. Sin embargo también puede provocar desafíos de seguridad ya que no se tiene la información

necesaria de todos los artefactos. Un solo componente comprometido puede ser un foco de ataques.

Cómo se produce un ataque a la cadena de suministro

Cualquier organización que utilice *software* de terceros o subcontraten colaboradores externos, establece relaciones de confianza, que forma parte de la cadena de suministro. Una vez que el atacante consigue violar la seguridad y acceder a la red de un colaborador, puede saltar a la red de la organización y extender el ataque.

Es frecuente que los proveedores de servicios administrados (MSP) sean objetivos habituales, ya que suelen tener ciertos permisos en las redes de los clientes y sus entornos *cloud*. Los MSP ofrecen a los atacantes un escaparate de accesos que pueden explotar de manera más sencilla que hacerlo de manera directa.

Algunos ejemplos de ataques recientes:

- SolarWinds en 2020 [73]
- Kaseya [74]
- CodeCov en 2021 [75]

8. Caso práctico

En este capítulo, se pone en práctica un ejemplo de cómo aplicar la capa de seguridad que aplica DevSecOps durante el desarrollo de un proyecto. A continuación, se expone un resumen del mismo. Si se desea consultar el proceso completo, se encuentra disponible en la sección de anexos -exactamente en el anexo 4- presente en el punto 11 de este trabajo.

En concreto, se va a utilizar un proyecto público -accesible para todo el mundo- desde **GitHub** [76]. Una vez importado, se le añadirá al código una dependencia asociada a la biblioteca **log4j de Java** [77]. A esta biblioteca se le ha detectado recientemente una vulnerabilidad, la cual, provoca una brecha de seguridad que afecta al servicio web Apache. Por lo que han saltado las alarmas en todo el mundo [78].

Una vez importado el proyecto y añadida la vulnerabilidad al código, se hace uso del protocolo **OAuth** [79] que permite compartir información de sitios web o aplicaciones desde un origen (proveedor) a un destino (consumidor). En este caso se realiza un OAuth desde una cuenta de GitHub (donde se encuentra el proyecto) para que pueda tener acceso a la información dos plataformas que ofrecen su servicio web, **Synk** [80] -en búsqueda de código malicioso- y **SonarCloud** [81] -revisión y análisis estático del código- (herramientas mencionadas en el apartado 7.6).

Proyecto de GitHub

El código del proyecto elegido para el caso práctico se encuentra en el siguiente enlace, almacenado en un repositorio de GitHub y que será utilizado como el desarrollo al que se le aplican las herramientas de seguridad.

Integración de Snyk con GitHub

Como se ha comentado anteriormente, se debe autorizar a Snyk para que tenga acceso a la información que contiene el proyecto de Github a través del protocolo OAuth. Una vez seleccionados los permisos y acciones a los que se le da acceso a la herramientas Snyk (siguiendo las indicaciones de la web de la propia herramienta).

Una vez se finaliza la configuración, se consigue que ambas plataformas -GitHub y Snyk- puedan realizar conexiones seguras e intercambiar información entre ellas. A continuación, se utilizan las aplicaciones de escritorio de GitHub y de Visual Studio Code, para poder editar el código del proyecto, con el objetivo de añadir la librería comprometida con la vulnerabilidad log4j.

En concreto, este es el fragmento de código que se añade al proyecto.

```
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
  <version>2.14.1</version>
</dependency>
```

Figura 17 Inserción dependencia log4j V14 bis

Una vez se lanzan las tareas de Snyk en busca de vulnerabilidades en el código del proyecto, se puede observar que ha detectado la vulnerabilidad log4j, introducida de manera intencionada en el proyecto con anterioridad, que afecta al servidor web Apache.

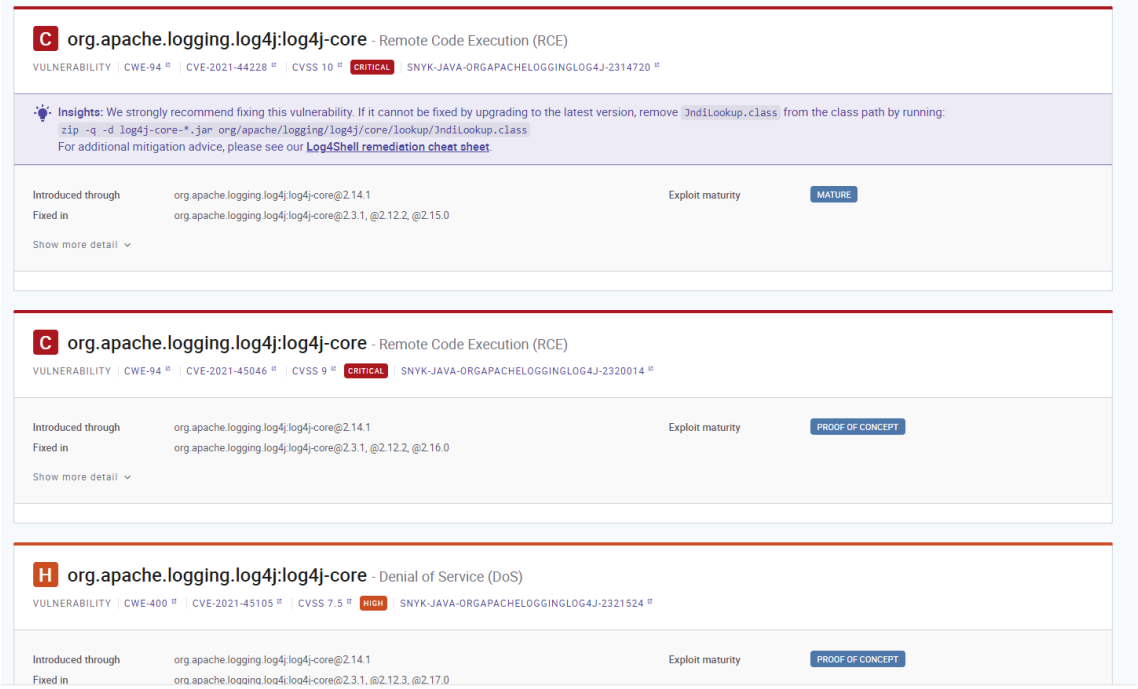


Figura 18 Detección vulnerabilidad Snyk bis

Integración de SonarCloud con GitHub

Los pasos son muy parecidos a los que se han seguido con Snyk. Se debe realizar el asistente para dar autoridad a SonarCloud para que acceda a la información de los proyectos de GitHub.

Para realizar los análisis del código de GitHub, SonarCloud solicita realizar dos acciones. La primera generar un *token* para utilizar como secreto durante las conexiones seguras entre ambas plataformas. Básicamente consiste en generar un *token* que compartirán ambas plataformas para la conexión segura y añadir al código la url de SonarCloud para tramitar la información.

Una vez realizada la configuración, desde la web GitHub, se pueden observar las tareas que se están realizando sobre el código del proyecto.

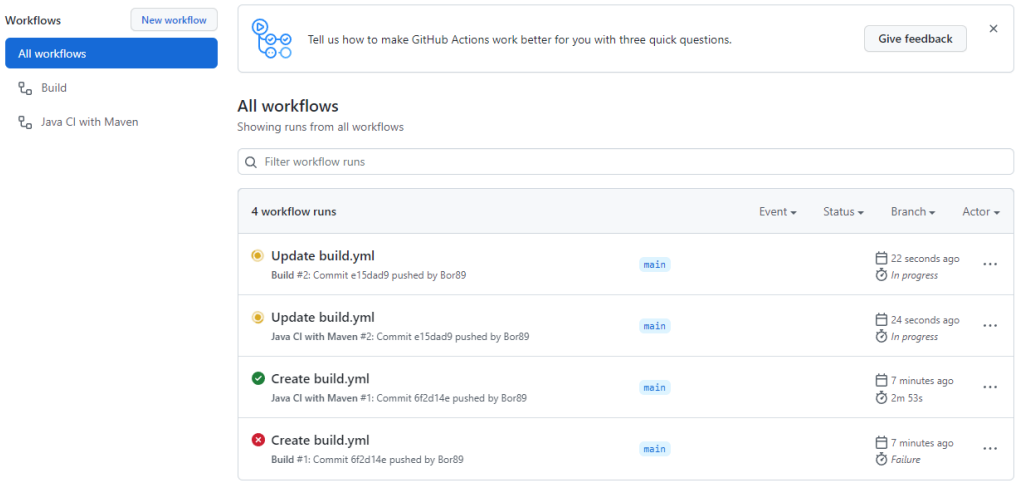


Figura 19 Tareas 2 bis

Finalmente, se pueden ver todos los test que se han configurado y que en este momento, se encuentran analizando el código del proyecto.

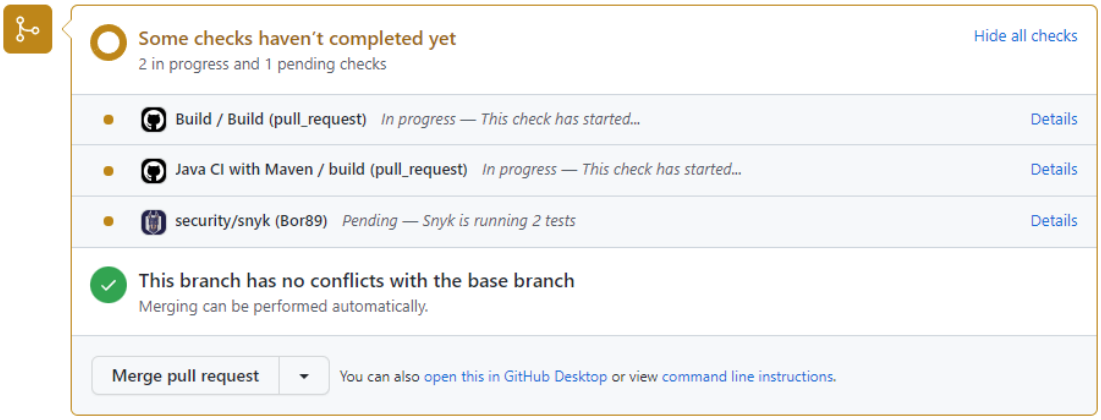


Figura 20 Test GitHub bis

Donde Snyk se ocupa de las vulnerabilidades de seguridad y SonarCloud de los problemas en la escritura del código.

Resultados del caso práctico

Se ha podido comprobar algunas de las características que ofrece DevSecOps. Como son las operaciones automatizadas que han realizado las herramientas. Mientras se desarrolla el código y avanza el proyecto, las herramientas realizan sus análisis en tiempo real, ofreciendo reportes e información muy valiosa que ayuda no solo a los desarrolladores, sino a todos los equipos participantes, como el de operaciones o seguridad, a subsanar muchos problemas antes de seguir avanzando con el CI/CD.

También se ha podido ver su variabilidad y adaptación en diferentes entornos, ya que las herramientas ofrecen sus servicios en diferentes plataformas, y son capaces de analizar tanto código propio como elementos de terceros, a la vez que se puede configurar la sensibilidad de la detección de errores. La importancia de aplicar la seguridad desde los inicios del proyecto queda patente para no arrastrar errores en etapas posteriores. En este ejemplo no se han producido falsos positivos, pero habría que tenerlo en cuenta para futuras ocasiones.

Las herramientas utilizadas como Snyk, de tipo SaaS, se adapta perfectamente al entorno CI/CD -como de GitHub- evaluando el código de una manera automatizada y reparando las posibles vulnerabilidades localizadas en las dependencias del código, para que los desarrolladores den una solución de manera rápida y eficaz.

SoncarCloud y su análisis para detectar irregularidades en la calidad del código del proyecto, asegura el continuo mantenimiento y su confianza. Junto a Snyk, han formado una buena combinación.

9. Conclusiones

Es innegable que el presente y el futuro de la sociedad actual, se basa en un entorno conectado, donde los datos y la información han pasado a tener un papel tan importante, que en muchas ocasiones su valor es mayor que el dinero. Incluso, su pérdida o sustracción puede situar a las compañías en una situación muy comprometida.

No es de extrañar que durante los últimos años, cada vez sean más frecuentes los casos en que grandes empresas sufren ciberataques que ponen en riesgo su información. En muchas ocasiones, las medidas de seguridad tomadas no son adecuadas para contener a los *hackers*, cada vez están más especializados.

Para poder interactuar con este mundo conectado, la mayoría de las personas utiliza dispositivos con acceso a Internet, ya sean desde ordenadores, *smartphones*... o nuevos elementos que se han unido gracias al IOT (Internet de la cosas), como pueden ser coches, electrodomésticos, etc. Todos estos dispositivos, cuentan con aplicaciones (*software*) que facilitan a sus usuarios realizar tareas específicas que por ocio o trabajo, utilizan en el día a día.

Parece imposible que con este contexto, se pueda encontrar desarrollos de *software* en el que la seguridad no es una de sus principales inquietudes, pero nada más lejos, ya que dar de lado la seguridad, es una práctica más habitual de lo que se pueda pensar. Esta situación se da en la actualidad por arrastrar metodologías antiguas que no se adaptan a las necesidades actuales, junto la mala idea preconcebida que tienen muchos desarrolladores, de que aplicar seguridad a sus proyectos generan retrasos en sus entregas.

No solo es necesario trabajar con metodologías modernas de métodos ágiles e introducir la seguridad de manera forzada. Se debe realizar un cambio de mentalidad, para que todos los actores que intervienen en el desarrollo del *software*, asimilen la nueva forma de trabajar y se sientan parte de ella. Es decir, extender una filosofía de trabajo que sea aceptada por todos, se extingan los silos, la colaboración y comunicación sea continua y la seguridad se aplique de manera innata durante todo el desarrollo del proyecto.

Con el ejemplo práctico presentado en el tema 8, se ha podido ver en acción alguna herramienta comentada en la teoría. Y como conclusión, es asombroso comprobar cómo estas herramientas trabajan en tiempo real mientras avanza el desarrollo del proyecto. La implantación de estas herramientas parecen indispensables gracias a todo los beneficios que aportan, junto a la transparencia e independencia con la que trabajan.

Se puede afirmar que se han alcanzado los objetivos que demuestran todos los beneficios que aporta una metodología DevSecOps para el desarrollo de *software* seguro. Es cierto que solo se ha trabajado de manera práctica con pocas herramientas.

Sin embargo, se ha podido captar tu eficiencia y virtudes, y abre un mundo nuevo para descubrir toda su potencia y capacidad.

La metodología ágil seguida durante el desarrollo del presente trabajo ha sido acertada, ya que se han realizado cambios a lo largo del mismo, teniendo que realizar modificaciones en parte del contenido. Incluso se ha tenido que variar la planificación inicial para adaptar el trabajo hacia el resultado final esperado, que en un principio era abierto y no tenía nada cerrado de manera definitiva.

La introducción del caso práctico ha aportado credibilidad a toda la parte teórica expuesta, ya que se ha podido demostrar las capacidades y parte de la metodología DevSecOps en un CI/CD desde un proyecto en GitHub.

Como **líneas de trabajo futuro**, existe un abanico enorme de posibilidades, ya que en el presente trabajo se han presentado las ideas principales de las nuevas metodologías basadas en la seguridad durante el desarrollo. Y como se ha comentado anteriormente, solo que han puesto en práctica un par de herramientas, cuando existen una gran cantidad de ellas de diferentes índoles y competencias. Hay todo un mundo por explorar.

10. Glosario

- **Acoplamiento** Forma y nivel de interdependencia entre módulos de *software*; una medida de qué tan cercanamente conectados están dos rutinas o módulos de *software*.

- **Algoritmo** Conjunto de instrucciones definidas, ordenadas y acotadas para resolver un problema o realizar una tarea.

- **Amenaza** Toda acción que aprovecha una vulnerabilidad para atacar o invadir un sistema informático.

- **API** *Application Programming Interface*. Conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro *software* como una capa de abstracción.

- **Aplicaciones distribuidas** Aplicación con distintos componentes que se ejecutan en entornos separados, normalmente en diferentes plataformas conectadas a través de una red.

- **Artefacto** Producto tangible resultante del proceso de desarrollo de *software*.

- **Base de Datos** Conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

- **Buenas prácticas** Toda experiencia que se guía por principios, objetivos y procedimientos apropiados o pautas aconsejables que se adecúan a una determinada perspectiva normativa o a un parámetro consensuado, así como también toda experiencia que ha arrojado resultados positivos, demostrando su eficacia y utilidad en un contexto concreto.

- **Build** Proceso de compilación y ejecución de un código.

- **Cadena de suministro** Conjunto de actividades, instalaciones y medios de distribución necesarios para llevar a cabo el proceso de venta de un producto en su totalidad.

- **CD/CI** Método para distribuir aplicaciones a los clientes con frecuencia mediante el uso de la automatización en las etapas del desarrollo de aplicaciones.

- **Cibercrimen** El cibercrimen es una actividad delictiva dirigida a un ordenador, una red informática o un dispositivo en red, o bien utiliza uno de estos.

- **Ciberseguridad** (Seguridad informática) Práctica de defender las computadoras, los servidores, los dispositivos móviles, los sistemas electrónicos, las redes y los datos de ataques maliciosos.

- **CLASP** Metodología diseñada para insertar seguridad en cada fase del ciclo de vida.

- **Cloud** Red mundial de servidores, cada uno con una función única. La nube no es una entidad física, sino una red enorme de servidores remotos de todo el mundo que están conectados para funcionar como un único ecosistema.

- **Cloud Computing** La computación en la nube es la entrega de diferentes servicios a través de Internet. Estos recursos incluyen herramientas y aplicaciones como almacenamiento de datos, servidores, bases de datos, redes y *software*.

- **Clúster** Conjuntos o conglomerados de ordenadores construidos mediante la utilización de *hardware* comunes y que se comportan como si fuesen una única computadora.

- **Código fuente** Archivo o conjunto de archivos, que contienen instrucciones concretas, escritas en un lenguaje de programación, que posteriormente compilan uno o varios programas.

- **Contenedores** Los contenedores son unidades ejecutables de *software* donde se empaqueta el código de aplicación, junto con sus bibliotecas y dependencias, de forma común para que se pueda ejecutar en cualquier lugar, ya sea en el escritorio, en la TI tradicional o en el cloud.

- **Cuadrante mágico de Gartner** Recoge la culminación de la investigación de un mercado específico, y proporciona una visión panorámica de las posiciones relativas de sus competidores.

- **DevOps** Es una combinación de los términos ingleses *development* (desarrollo) y *operations* (operaciones), designa la unión de personas, procesos y tecnología para ofrecer valor a los clientes de forma constante.

- **DevSecOps** Proceso de integración de la seguridad en los procesos de desarrollo y operaciones de aplicaciones informáticas.

- **Diagrama de Gantt** Herramienta útil para planificar proyectos.

- **Distribuido** Servicio que se encuentra localizado en varios entornos y en diferentes localizaciones físicas.

- **DNS** *Domain Name System*. Su función más importante es "traducir" nombres inteligibles para las personas en identificadores binarios asociados con los equipos conectados a la red, esto con el propósito de poder localizar y direccionar estos equipos mundialmente.

- ***Dumb Terminal*** Equipo informático sin capacidad de procesamiento de datos independiente.

- ***E-commerce*** Proceso de compra y venta de productos o servicios en internet.

- **Enmascaramiento de datos** Proceso mediante el cual se cambian ciertos elementos de los datos de un almacén de datos, cambiando su información pero consiguiendo que la estructura permanezca similar, de forma que la información sensible quede protegida.

- **Escalabilidad** Medida de la eficacia del crecimiento de un equipo, servicio o aplicación para satisfacer las exigencias de aumento del rendimiento.

- **Escritorio virtual** Acceso a un computador, cuyo poder de procesamiento no se encuentra en un PC físico, sino en servidores ubicados en un *datacenter*.

- **Falso negativo** Cuando un antivirus no detecta un *malware*, cuando deja pasar o ejecutar un código malicioso en el sistema que está protegiendo.

- **Falso positivo** Detección de un archivo como virus (o alguna otra clase de *malware*) por parte de un antivirus, cuando en realidad no es ningún virus o *malware*.

- ***Feedback*** Adquisición de información de los clientes, sobre un producto, la calidad del servicio de una marca, etc.

- ***Firewall*** También conocido como cortafuegos, es un elemento informático que trata de bloquear el acceso, a una red privada conectada a Internet, a usuarios no autorizados.

- ***Framework*** Marco o esquema de trabajo generalmente utilizado por programadores para realizar el desarrollo de *software*. Utilizar un *framework* permite agilizar los procesos de desarrollo ya que evita tener que escribir código de forma repetitiva, asegura unas buenas prácticas y la consistencia del código.

- ***Fuzzing*** Técnica de pruebas de *software*, a menudo automatizado o semiautomatizado, que implica proporcionar datos inválidos, inesperados o aleatorios a las entradas de un programa de ordenador. Entonces se monitorizan las excepciones tales como caídas, aserciones de código erróneas, o para encontrar potenciales filtraciones de memoria.

- **GitHub** Plataforma de alojamiento, propiedad de Microsoft, que ofrece a los desarrolladores la posibilidad de crear repositorios de código y guardarlos en la nube de forma segura, usando un sistema de control de versiones, llamado Git.

- **Hacker ético** Son personas contratadas para hackear un sistema e identificar y reparar posibles vulnerabilidades, lo que previene eficazmente la explotación por hackers maliciosos.

- **Hacking** Búsqueda y explotación de vulnerabilidades de seguridad en sistemas o redes.

- **Host** Computadoras u otros dispositivos (tabletas, móviles, portátiles) conectados a una red que proveen y utilizan servicios de ella.

- **Idle** Integrated Development and Learning Environment. Intérprete de código de lenguajes de programación.

- **Imagen** Archivo compuesto por múltiples capas, que se utiliza para ejecutar código en un contenedor. Estas imágenes son las plantillas base desde la que se parte, ya sea para crear una nueva imagen o crear nuevos contenedores para ejecutar las aplicaciones.

- **Ingeniería social** Ataques que engañan al usuario para que realice una acción que permita instalar algún *malware* en un equipo.

- **Interfaz** Conexión funcional entre dos sistemas, programas, dispositivos o componentes de cualquier tipo, que proporciona una comunicación de distintos niveles, permitiendo el intercambio de información.

- **IOT** *Internet Of Things*. Relación entre algunos objetos con la conexión a internet.

- **Malware** Término amplio que describe cualquier programa o código malicioso que es dañino para los sistemas.

- **Maven** Herramienta de *software* para la gestión y construcción de proyectos Java, basado en formato XML.

- **MDM** *Mobile Device Manager*. *Software* que permite asegurar, monitorizar y administrar dispositivos móviles sin importar el operador de telefonía o proveedor de servicios.

- ***Merge Hell*** Cuando el repositorio de trabajo se vuelve muy diferente de las líneas de base definidas por los desarrolladores.

- **Metodología** Conjunto de métodos coherentes y relacionados por unos principios comunes.

- **MFA** Acceso Multi Factor. Contar con más de una manera de autenticación sobre un sistema.

- **Microservicio** Elementos independientes en los que se puede dividir una aplicación.

- ***Middleware*** *Software* que se sitúa entre un sistema operativo y las aplicaciones que se ejecutan en él. Básicamente, funciona como una capa de traducción oculta para permitir la comunicación y la administración de datos en aplicaciones distribuidas.

- **Modelo ágil** El enfoque ágil para el desarrollo de *software* busca distribuir de forma permanente sistemas de *software* en funcionamiento diseñados con iteraciones rápidas.

- **OASAM** *Open Android Security Assessment Methodology*. *Framework* que tiene por objetivo ser una metodología de análisis de seguridad de aplicaciones Android.

- ***On-premise*** Instalaciones propias o *in situ*. Se refiere a la utilización de servidores y entorno informático propios de la empresa.

- **OWASP** Proyecto de código abierto para el desarrollo de aplicaciones web seguras.

- **P2P** *Peer to Peer*, protocolo para compartir información de manera descentralizada.

- **Paradigma** Manera, estilo o forma de pensar y actuar, como por ejemplo, en la programación.

- **Parche** Diferentes cambios que se han aplicado a un programa para corregir errores, actualizarlo, eliminar secciones antiguas de *software* o simplemente añadirle funcionalidad.

- **Phishing** Engaño mediante el cual terceros pretenden que los usuarios compartan datos confidenciales.

- **Pipeline CD/CI** Conjunto completo de procesos que se ejecutan cuando se activa el trabajo en los proyectos. Las canalizaciones abarcan los flujos de trabajo, que coordinan los trabajos, y todo esto se define en el archivo de configuración de cada proyecto.

- **Proxy** Servidor, programa o dispositivo que hace de intermediario en las peticiones de recursos que realiza un cliente (A) a otro servidor (C).

- **REST** *Representational State Transfer*. Estilo de arquitectura *software* para sistemas hipermedia distribuidos como la *World Wide Web*.

- **Robusto** *Software* que se comporta de forma razonable en circunstancias que no fueron anticipadas, como un ataque o una falla del sistema en la especificación de requerimientos.

- **Rol** Colección de permisos definida para todo el sistema que se pueden asignar a usuarios específicos en contextos diferentes.

- **Scrum** *Framework* que permite trabajar en una serie de interacciones en equipo.

- **SDLC** *Systems Development Life Cycle*, Ciclo de desarrollo de sistemas. Proceso de creación o modificación de los sistemas, modelos y metodologías que se utiliza para desarrollar sistemas de *software*.

- **Security by design** Enfoque de garantía de la seguridad que formaliza el diseño, automatiza los controles de seguridad y simplifica las auditorías.

- **Seguridad de orquestación** Herramientas y soluciones que pueden trabajar juntas, comunicarse, compartir y exportar datos de una manera intuitiva y fácil, sin interrumpirse ni cancelarse entre sí, y agilizar el proceso de seguridad que permite que cada herramienta se utilice al máximo potencial.

- **Seguridad proactiva** Métodos que se utilizan para prevenir ciberataques.

- **Servicio** Conjunto de actividades que buscan responder a las necesidades de un cliente (*host*).

- **Servidor** Un servidor es un equipo informático que forma parte de una red y provee servicios a otros equipos cliente.

- **Silo** Incapacidad de algunas empresas para trabajar eficientemente entre departamentos o todas las áreas que la conforman, a consecuencia de utilizar más aplicaciones de las necesarias o cuentas con exceso de información almacenada.

- **Sistemas heredados** Sistema informático (equipos informáticos o aplicaciones) que ha quedado anticuado pero que sigue siendo utilizado por el usuario (generalmente, una organización o empresa) y no se quiere o no se puede reemplazar o actualizar de forma sencilla.

- **SLA** *Service Level Agreement*. Contrato que describe el nivel de servicio que un cliente espera de su proveedor. En español, también se llama Acuerdo de Nivel de Servicio (ANS).

- ***Software*** Conjunto de los componentes lógicos necesarios que hace posible la realización de tareas específicas, en contraposición a los componentes físicos que son llamados *hardware*.

- ***Software* de terceros** Las aplicaciones de terceros son programas escritos para trabajar internamente con sistemas operativos, pero son escritos por personas o compañías aparte de los propios miembros de la compañía que provee de este sistema operativo.

- ***Software* libre** Modelo de desarrollo de *software* basado en la colaboración abierta. Se enfoca en los beneficios prácticos (acceso al código fuente) y en cuestiones éticas o de libertad que tanto se destacan en el *software* libre.

- ***Sprint*** Miniproyecto de no más de un mes (ciclos de ejecución muy cortos,entre una y cuatro semanas), cuyo objetivo es conseguir un incremento de valor en el producto que se está construyendo.

- **SQL** *Structured Query Language*. Lenguaje de programación que te permite manipular y descargar datos de una base de datos.

- ***Tester*** Planifican y llevan a cabo pruebas de *software* de los ordenadores para comprobar si funcionan correctamente.

- **TIC** Tecnologías que utilizan la informática, la microelectrónica y las telecomunicaciones para crear nuevas formas de comunicación a través de herramientas de carácter tecnológico y comunicacional, esto con el fin de facilitar la emisión, acceso y tratamiento de la información.

- ***Token*** Referencia (un identificador) que regresa a los datos sensibles a través de un sistema de tokenización.

- **Usabilidad** Capacidad de un *software* de ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso.

- **Vector de ataque** Formas o medios que permiten a ciberdelincuentes transmitir códigos maliciosos, con el propósito de obtener beneficios económicos.

- **Virtualización** Entorno informático simulado, o virtual, que sustituye a un entorno físico.

- **Vulnerabilidad** Fallo de seguridad en un programa o sistema informático.

- ***Workflows*** Flujos de trabajo, automatización de las tareas de modo que quedan debidamente jerarquizadas, organizadas y automatizadas.

- **YAML** Formato de serialización de datos que significa YAML no es lenguaje de marcado.

11. Bibliografía

- [1] Decálogo hacia la ciberseguridad [Internet] Incibe.es. Consultado el 21 de septiembre. Disponible en Enlace
- [2] ¿Qué es DevSecOps? [Internet]. Redhat.com. Consultado el 21 de septiembre de 2021, Disponible en Enlace
- [3] SecDevps. [Internet]. Elladodelmal.com. Consultado 23 de septiembre de 2021, Disponible en Enlace
- [4] Desarrollo seguro. [Internet] Ciberseguridad.com. Consultado el 24 de septiembre de 2021. Disponible en Enlace
- [5] ¿Qué es la seguridad de las aplicaciones? [Internet] Vmware.com. Consultado el 26 de septiembre de 2021. Disponible en Enlace
- [6] Las 4 claves de la seguridad de la información [Internet] universidadviu.com. Fecha de publicación [18 de abril de 2018] Consultado el 05 de diciembre de 2021. Disponible en Enlace
- [7] Recomendaciones de Seguridad para Autenticación Multi-Factor [Internet/PDF] ccn.cni.es. Consultado el 05 de diciembre de 2021. Disponible en Enlace
- [8] Cifrado de la información [Internet]. Fecha de publicación [2020] Consultado el 05 de diciembre de 2021. Disponible en Enlace
- [9] ¿Qué es la seguridad de las aplicaciones? [Internet] Vmware.com. Consultado el 26 de septiembre de 2021. Disponible en Enlace
- [10] ¿Cómo elaborar un "Estado del Arte"? [Internet] YouTube. Consultado el 25 de septiembre de 2021. Disponible en Enlace
- [11] Desarrollo seguro de aplicaciones para dispositivos móviles. [Internet] Incibe-Cert.es. Consultado el 25 de septiembre de 2021. Disponible en Enlace
- [12] TIC (Tecnologías de la Información y Comunicación) [Internet]. Wikipedia. Consultado el 04 de octubre de 2021. Disponible en Enlace
- [13] Ciberseguridad, retos y amenazas a la seguridad nacional en el ciberespacio [Internet/PDF] ieee.es. Autor: María José Caro Bejarano. Página 61. Consultado el 05 de diciembre de 2021. Disponible en Enlace
- [14] La importancia de una estrategia de ciberseguridad [Internet]. Incibe. Consultado el 06 de octubre de 2021. Disponible en Enlace
- [15] Arquitectura cliente-servidor [Internet]. cio-wiki.org. Consultado el 08 de octubre de 2021. Disponible en Enlace
- [16] Arquitectura *Peer to Peer* (P2P) [Internet]. reactiveprogramming.io. Consultado el 08 de octubre de 2021. Disponible en Enlace
- [17] Arquitectura aplicación web [Internet]. afahc.ro. Consultado el 08 de octubre de 2021. Disponible en Enlace
- [18] Arquitecturas de aplicaciones [Internet]. redhat.com. Fecha de publicación [9 de marzo de 2020] Consultado el 05 de diciembre de 2021. Disponible en Enlace

- [19] Arquitectura *cloud* [Internet]. vmware.com. Consultado el 10 de octubre de 2021. Disponible en Enlace
- [20] Lado oscuro de los aplicativos cliente-servidor [Internet]. empresas.blogthinkbig.com. Consultado el 05 de diciembre de 2021. Disponible en Enlace
- [21] Servicios SAAS: una buena práctica en ciberseguridad [Internet]. drag.es. Fecha de publicación [03 de febrero de 2021] Consultado el 05 de diciembre de 2021. Disponible en Enlace
- [22] Metodología [Revista]. Autores: Rivas, Carlos Ignacio, Corona, Verónica Paola, Gutiérrez, José Fructuoso y Hernández, Lizeth. Título: Metodologías actuales de desarrollo de software. Nombre: Revista Tecnología e Innovación, pág. 982, No.5 Vol.2 diciembre 2015. Consultado el 11 de octubre de 2021. Disponible en Enlace
- [23] Modelo de desarrollo en cascada [Internet]. comparaSoftware. Consultado el 11 de octubre de 2021. Disponible en Enlace
- [24] Modelo en cascada [Internet]. uniWebSidad. Consultado el 11 de octubre de 2021. Disponible en Enlace
- [25] Pros y contras de la metodología en cascada [Internet]. obsBussinesSchool. Consultado el 11 de octubre de 2021. Disponible en Enlace
- [26] *Secure Software* [Internet]. securesoftware.com. Consultado el 11 de octubre de 2021. Disponible en Enlace
- [27] [Paper científico]. Autores: Hala Assal, Sonia Chiasson. Título: Metodologías actuales de desarrollo de software. Nombre: Security in the Software Development Lifecycle Fe de pblicación [12-14 de agosto de 2018]. Consultado el 14 de octubre de 2021. Disponible en Enlace
- [28] How to approach secure software development [Internet]. ptsecurity.com. Consultado el 14 de octubre de 2021. Disponible en Enlace
- [29] *CLASP Process* [Internet]. us-cert.cisa.gov. Consultado el 20 de octubre de 2021. Disponible en Enlace
- [30] *Secure Software Development Framework SSDF* [Internet]. csrc.nist.gov. Consultado el 20 de octubre de 2021. Disponible en Enlace
- [31] *Secure SDLC Best Practices* [Internet]. snyk.io. Consultado el 18 de octubre de 2021. Disponible en Enlace
- [32] Ciclo de vida de desarrollo de *software* seguro [Internet]. linkedin.com. Consultado el 18 de octubre de 2021. Disponible en Enlace
- [33] Mitre Corp [Internet]. mitre.org. Consultado el 16 de octubre de 2021. Disponible en Enlace
- [34] Mitre Corp [Internet]. forbes.com.mx. Consultado el 16 de octubre de 2021. Disponible en Enlace
- [35] *2021 CWE Top 25 Most Dangerous Software Weaknesses* [Internet]. mitre.org. Consultado el 16 de octubre de 2021. Disponible en Enlace
- [36] *2021's Most Dangerous Software Weaknesses* [Internet]. threatpost.com. Consultado el 16 de octubre de 2021. Disponible en Enlace
- [37] *Top threat modeling frameworks* [Internet]. resources.infosecinstitute.com. Consultado el 16 de octubre de 2021. Disponible en Enlace
- [38] *Secure SDLC Best Practices* [Internet]. snyk.io. Consultado el 18 de octubre de 2021. Disponible en Enlace
- [39] *DevOps* [Internet]. Jetbrains.com. Consultado el 30 de octubre de 2021. Disponible en Enlace
- [40] Ley Orgánica 3/2018 Protección de Datos Personales y garantía de los derechos digitales [Internet]. BOE.es. Consultado el 01 de noviembre de 2021. Disponible en Enlace

- [41] Qué es *DevOps* [Internet]. dynatrace.com. Consultado el 05 de octubre de 2021. Disponible en Enlace
- [42] *DevOps tools* [Internet]. aquasec.com. Consultado el 08 de octubre de 2021. Disponible en Enlace
- [43] *DevOps Concepts and Challenges* [Artículo técnico]. Autores: Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojocic, Paulo Meirelles. Título: *A Survey of DevOps Concepts and Challenges*. . Fecha de publicación [18 de noviembre de 2019] Consultado el 09 de noviembre de 2021. Disponible en Enlace
- [44] *DevOps* [Paper científico]. Autores: Luz Welder, Gustavo Pinto, Bonifacio Rodrigo. Título: *Adopting DevOps in the Real World: A Theory, a Model, and a Case Study*. Fecha de publicación [25 de junio de 2019] Consultado el 10 de noviembre de 2021. Disponible en Enlace
- [45] *elsevier.com* [Artículo revista]. Autores: Alok Mishra, Ziadoon Otaiwi. Título: *DevOps and software quality: A systematic mapping*. Nombre revista: Elsevier. Computer Science Review 38 (2020). Fecha de publicación [03 de octubre de 2020]. Consultado el 10 de noviembre de 2021. Disponible en Enlace
- [46] DevOps CI. [Internet] deloitte.com. Consultado el 18 de octubre de 2021. Disponible en Enlace
- [47] *Merge Hell*. [Internet] redhat.com. Consultado el 18 de octubre de 2021. Disponible en Enlace
- [48] Secuencia escalonada azul-verde. [Internet] thenewstack.io. Consultado el 19 de octubre de 2021. Disponible en Enlace
- [49] Definición DevSecOps [Internet]. dynatrace.com. Autor: Jack Marsal. Fecha de publicación el 20 de julio de 2021. Consultado el 08 de noviembre de 2021. Disponible en Enlace
- [50] DevSecOps [Internet]. dynatrace.com. Consultado el 09 de noviembre de 2021. Disponible en Enlace
- [51] DevSecOps, elementos esenciales [Internet]. aquasec.com. Consultado el 12 de noviembre de 2021. Disponible en Enlace
- [52] *5 Keys to a Secure DecOps Workflow*, [Internet] dig.sysdig.com. Consultado el 12 de noviembre de 2021. Disponible en Enlace
- [53] SecDevOps, buenas prácticas de seguridad desde la dase de desarrollo [Internet] izertis.com. Consultado el 14 de noviembre de 2021. Disponible en Enlace
- [54] Beneficios y mejores prácticas de SecDevOps: no hay agilidad sin seguridad. [Internet] crashtest-security.com. Consultado el 14 de noviembre de 2021. Disponible en Enlace
- [55] DevSecOps [Artículo digital] ieee.org. Autor: Kim Carter. Título: Francois Raynaud on DevSecOps. Nombre revista: *The Software Egineering*. Fecha de publicación [septiembre-octubre de 2017]. Consultado el 15 de noviembre de 2021. Disponible en Enlace
- [56] *Focusing on the DevOps Pipeline*. [Internet] medium.com. Consultado el 16 de octubre de 2021. Disponible en Enlace
- [57] Falsos positivos. [Internet] redeszone.net. Consultado el 16 de octubre de 2021. Disponible en Enlace
- [58] Gartnet, *Magic Quadrant for Application Security Testing*. [Internet] gartner.com. Consultado el 16 de octubre de 2021. Disponible en Enlace
- [59] *GitHub*. [Internet] github.com. Consultado el 16 de octubre de 2021. Disponible en Enlace
- [60] Github actions. [Internet] github.com. Consultado el 17 de octubre de 2021. Disponible en Enlace
- [61] Introducción a YAML. [Internet] geekflare.com. Consultado el 17 de octubre de 2021. Disponible en Enlace

- [62] Trivy *open source scanner*. [Internet] cncf.io. Consultado el 17 de octubre de 2021. Disponible en Enlace
- [63] Starboard, *the kubernetes-native Sucurity Toolkit*. [Internet] aquasec.com. Consultado el 17 de octubre de 2021. Disponible en Enlace
- [64] OWASP ZAP. [Internet] owas.org. Consultado el 17 de octubre de 2021. Disponible en Enlace
- [65] Vault HashiCorp. [Internet] vaultproject.io. Consultado el 17 de octubre de 2021. Disponible en Enlace
- [66] SonarQube. [Internet] sonarqube.org. Consultado el 17 de octubre de 2021. Disponible en Enlace
- [67] Vulnerabilidades Open Source. [Internet] softzone.es. Consultado el 18 de diciembre de 2021. Disponible en Enlace
- [68] Seguridad continúa con Snyk. [Internet] itdo.com. Consultado el 18 de diciembre de 2021. Disponible en Enlace
- [69] SonarCloud. [Internet] sdos.es. Consultado el 18 de diciembre de 2021. Disponible en Enlace
- [70] Aqua Security. [Internet] aquasec.com. Consultado el 17 de octubre de 2021. Disponible en Enlace
- [71] SolarWinds. [Internet] solarwinds.com. Consultado el 17 de octubre de 2021. Disponible en Enlace
- [72] *Software Composition Analysis* (SCA) [Paper científico]. Autores: Luz Welder, Gustavo Pinto, Bonifacio Rodrigo. Título: *The Dynamics of Software Composition Analysis.* Fecha de publicación [30 de septiembre de 2019]. Consultado el 18 de noviembre de 2021. Disponible en Enlace
- [73] Ataque SolarWinds. [Internet] xataca.com. Consultado el 17 de octubre de 2021. Disponible en Enlace
- [74] Ataque Kaseya. [Internet] xataca.com. Consultado el 17 de octubre de 2021. Disponible en Enlace
- [75] Ataque CodeCov. [Internet] incibe-cert.es. Consultado el 17 de octubre de 2021. Disponible en Enlace
- [76] GitHub. [Internet] github.com. Consultado el 19 de diciembre de 2021. Disponible en Enlace
- [77] Vulnerabilidad log4j Apache. [Internet] incibe.es. Consultado el 19 de diciembre de 2021. Disponible en Enlace
- [78] log4j. [Internet] elespanol.es. Consultado el 19 de diciembre de 2021. Disponible en Enlace
- [79] OAuth. [Internet] wikipedia.org. Consultado el 19 de diciembre de 2021. Disponible en Enlace
- [80] Snyk. [Internet] snyk.io. Consultado el 20 de diciembre de 2021. Disponible en Enlace
- [81] SonarCloud. [Internet] sonarcloud.io. Consultado el 20 de diciembre de 2021. Disponible en Enlace
- [82] Kubernetes. [Internet] Kubernetes.io. Consultado el 18 de octubre de 2021. Disponible en Enlace
- [83] Contenedor de imágenes. [Internet] aquasec.com. Consultado el 18 de octubre de 2021. Disponible en Enlace
- [84] Técnicas de ataques en la nube. [Internet] redeszone.net. Consultado el 18 de octubre de 2021. Disponible en Enlace
- [85] Seguridad en la nube. [Internet] aquasec.com. Consultado el 18 de octubre de 2021. Disponible en Enlace
- [86] Planes Snyk. [Internet] snyk.io. Consultado el 23 de diciembre de 2021. Disponible en Enlace

- [87] GitHub Desktop. [Internet] github.com. Consultado el 23 de diciembre de 2021. Disponible en Enlace
- [88] Visual Studio Code. [Internet] visualstudio.com. Consultado el 23 de diciembre de 2021. Disponible en Enlace
- [89] SonarCloud. [Internet] sonarcloud.io. Consultado el 20 de diciembre de 2021. Disponible en Enlace
- [90] Maven. [Internet] wikipedia.org. Consultado el 23 de diciembre de 2021. Disponible en Enlace
- [91] DevSecOps y su función en la CD. [Internet] jetbrains.com Consultado el 05 de noviembre de 2021. Disponible en Enlace

Otros TFG y TFM consultados como referencia inicial.

- Ventajas e implementación de un sistema IDS/SIEM en el ámbito familiar. [UOC.edu/O2]. José Antonio Salom Martín. Consultado el 20 de septiembre de 2021. Disponible en Enlace
- Seguridad en *smartphones*, Análisis de riesgos y vulnerabilidades [UOC.edu/O2] Alfonso González Fernández. Consultado el 20 de septiembre de 2021. Disponible en Enlace
- Seguridad en Internet de las cosas. [UOC.edu/O2]. Francisco Montes Gallardo. Consultado el 20 de septiembre de 2021. Disponible en Enlace

12. Anexos

A1 Kubernetes

Es una herramienta de orquestación de Google que gestiona microservicios o apelaciones en contenedores en un clúster distribuido de nodos. Proporciona la infraestructura necesaria con escalado de los contenedores. Además, colabora para ocultar la complejidad del sistema ofreciendo APIs REST para gestionar los contenedores. A la vez que es multiplataforma (*cloud*, AmazonWS, Azure, Apache…) [82].

Ventajas de Kubernetes
Ha incrementado su popularidad gracias a sus experiencia en I+D y operaciones que ofrece Google, junto a la gran comunidad que ha ido gestando a su alrededor y que permite ser desplegado en un sinfín de escenarios distintos.

Componentes y arquitectura
Se podría pensar en cualquier otra infraestructura, pero la verdad es que sigue siendo cliente-servidor, con posibilidad de configurarlo en alta disponibilidad. El servidor maestro está formado por diferentes componentes que forman la infraestructura, almacenamiento, controlador Kube, controlador *cloud*, servidor DNS, proxy…

Componentes del nodo maestro

- Clúster etcd. Almacenamiento distribuido donde se almacena los datos del clúster de Kubernetes. A través de las API´s se activa la actualización de la información que contiene el nodo.
- *Kube-APIServer*. Servidor de la API. Gestiona todas las solicitudes REST que recibe el nodo, actúa como *frontend*.
- *Kube-controller manager*. Ejecuta procesos de controladores en segundo plano para regular el estado del clúster y realizar las tareas rutinarias.
- *Cloud-controller manager*. Gestiona los procesos del controlador que cuenta con dependencias otro proveedor de servicios *cloud* (como balancear la carga o los volúmenes)
- *Kube-scheduler*. Programa los *pods* (grupos de contenedores donde se ejecutan los procesos de las aplicaciones) basándose en la necesidad de los recursos de cada momento (por ejemplo, otorgando o reduciendo la capacidad de memoria o procesamiento que requiere la aplicación)

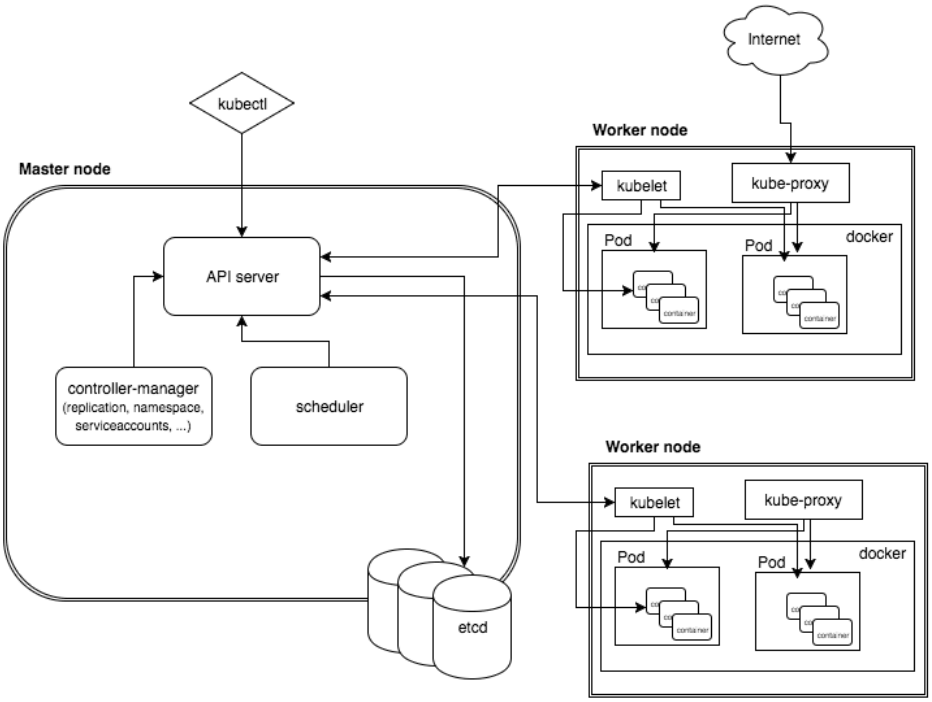


Figura 21 Esquema de nodo maestro y worker en Kubernetes

A2 Contenedor de imágenes

Es un archivo con código ejecutable capaz de crear un contenedor dentro de un sistema determinado. Una imagen no puede cambiarse y puede ser desplegada en cualquier entorno. Es el componente central de una arquitectura de contenedores.

Las imágenes contienen todo lo que necesita un contenedor para ejecutarse, un motor (como Docker), bibliotecas del sistemas, la configuración, etc. La imagen comparte el núcleo del sistema operativo del host, por este motivo necesita incluir el SO completo.

Una imagen de un contenedor está compuesta por diferentes capas, añadida a la imagen base. Estas capas permiten reutilizar los componentes y las configuraciones entre diferentes imágenes. Es importante crear las capas de manera óptima para reducir el tamaño del contenedor y mejorar el rendimiento [83].

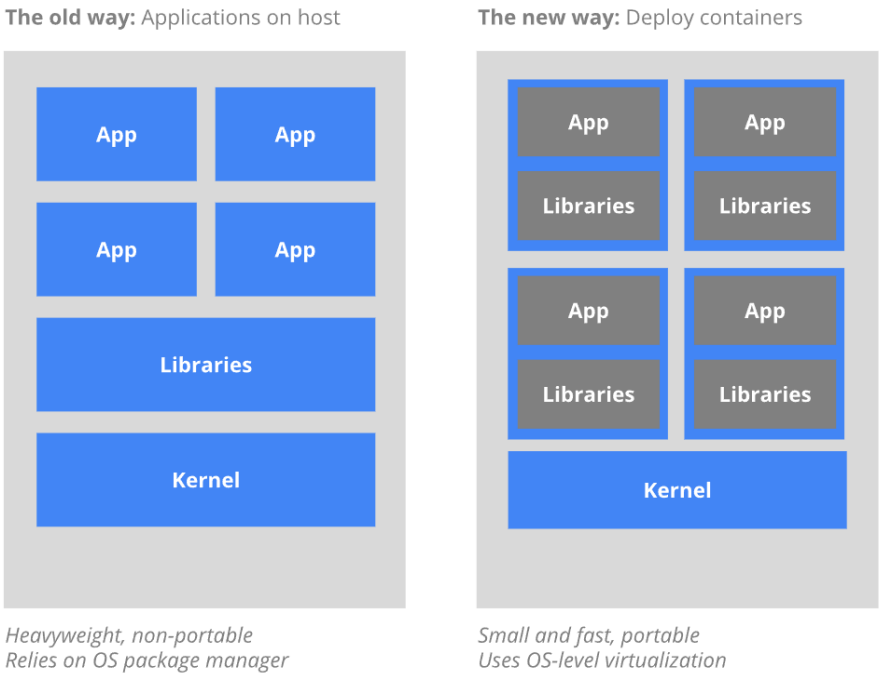


Figura 22 Esquema de contenedores

Imágenes padre y base

Imagen base. Se trata de una imagen de contenedor vacía que permite a los administradores crear imágenes desde cero.
Imagen padre. Es una imagen pre configurada que proporciona funcionalidades básicas, como un SSOO Linux, una BBDD MySQL o un sistemas de gestión como WordPress.

A pesar de estas distinciones, en muchas ocasiones se utilizan ambos términos indistintamente. Existen una gran cantidad de imágenes en contenedores públicos a disposición de los usuarios, como también lo están en repositorios privados, como Docker, AmazonWS o Azure.

A3 Seguridad en la nube

En la actualidad las organizaciones que quieren fomentar la colaboración e innovación, muestran su confianza en servicios de computación *cloud*.

¿Qué es la seguridad en la nube?
Las empresas y los gobiernos que desean fomentar la innovación y la colaboración confían cada vez más en los servicios de computación en la nube. En la actualidad, la gran mayoría de las empresas utilizan algún servicio en la nube y además, muchas de estas deciden almacenar información sensible. Aproximadamente un 20% de estas empresas han sufrido algún ataque en su infraestructura de nube pública [75].

La seguridad que se implementa en la nube, se basa en tecnología y procedimientos que protegen la infraestructura *cloud*. Para proteger datos y aplicaciones de las nuevas metodologías de ataques, se deben aplicar las mejores técnicas de seguridad y desarrollar nuevas estrategias que mejor se adapten al entorno particular de nube.

Figura 23 Nube

Buenas prácticas de seguridad en la nube
Revisión. Revisar las características de seguridad del entorno de igual manera que se probaría en un entorno *on premise*. Sobre todo para comprobar la calidad del *software* de terceros.

Gestión de identidades de acceso (IAM). Muy importantes en sistemas *cloud*, desde que los usuarios pueden hacer uso de los servicios desde cualquier lugar y dispositivo. Se puede supervisar su actividad y programar alertas en caso de comportamientos anómalos. Además, aporta autenticación multifactor (MFA) y características de inicio de sesión único (SSO).

Evitar la ingeniería social. Evitar el *pishing* y derivados de este es esencial, para ello se pueden tomar medidas como:

- Mostrar la importancia de no compartir con nadie las credenciales personales.
- Protección del correo electrónico.
- Exigir tiempos de espera de inicio de sesión y cambios de contraseñas periódicos.
- Programar alertas ante inicios de sesión desde lugares e ip´s diferentes.
- Uso de MFA obligatorio.

Auditar y optimizar las configuraciones
Debido a los continuos cambios que experimentan las infraestructuras *cloud*, se debe supervisar y verificar que las configuraciones aplicadas continúan siendo seguras. Cuando se escalan o replican cálculos o volúmenes de datos, pueden ocurrir errores de configuración que pueden comprometer la estabilidad de la seguridad [84].

A4 Desarrollo del caso práctico (Apartado 8)

Proyecto de GitHub

El código del proyecto elegido para el caso práctico se encuentra en el siguiente enlace (nombre del proyecto: *spring-petclinic*), que pertenece a un repositorio de GitHub. Al ser código libre, solo se requiere una cuenta de usuario de GitHub, para importar el código a esta cuenta y poder "colaborar" en su desarrollo. En este caso solo se utiliza a modo de ejemplo, y no se actualizará el proyecto principal.

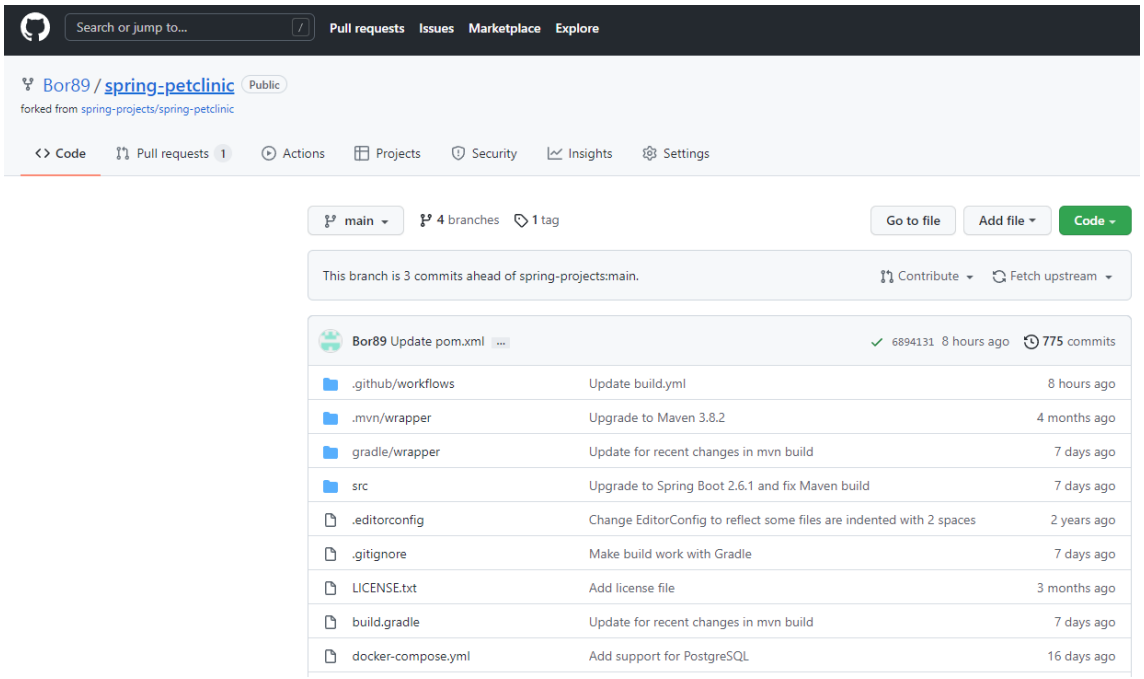


Figura 24 Proyecto pruebas

Integración de Snyk con GitHub

Es un servicio de pago pero dispone de una modalidad gratuita [86] con la que permite realizar test, perfecta para la ocasión. Una vez elegida la modalidad deseada, hay que seleccionar la plataforma a la que se quiere autorizar para que Snyk tenga acceso a la información, en este caso se selecciona GitHub.

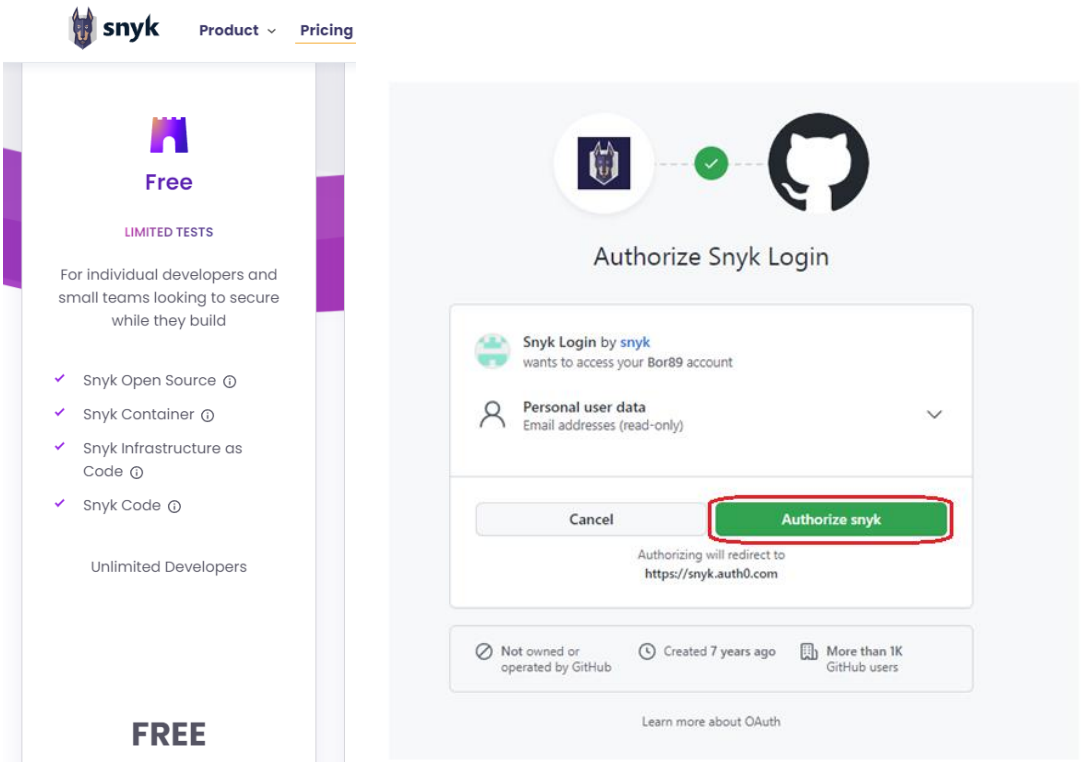


Figura 25 Plan Snyk y Autorización para GitHub

A continuación se seleccionan los proyectos a los que se le quiere dar permiso a Snyk.

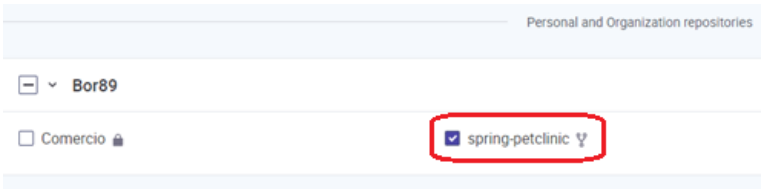


Figura 26 Acceso proyecto para Snyk

Se activan las opciones marcadas para indicar a Snyk que actúe con la máxima severidad y sensibilidad posible, es decir, que sea muy estricto y detecte el mayor número de vulnerabilidades posibles.

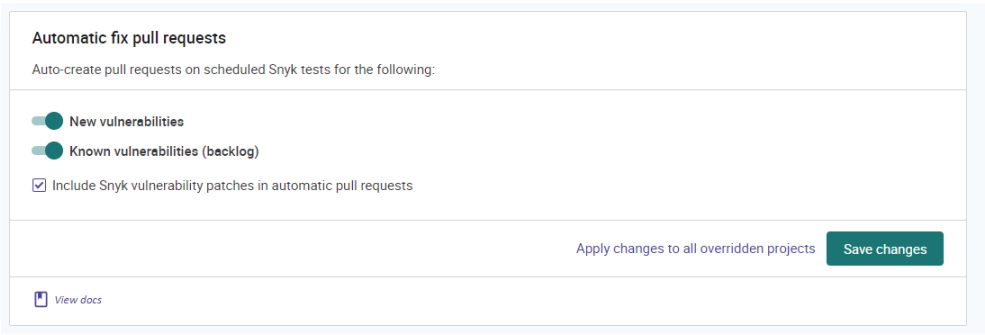


Figura 27 Severidad Snyk

Para que Snyk sea capaz de escanear los proyectos, se debe generar un *token* de acceso indicando los permisos que se le otorga. En un desarrollo profesional, se debe precisar la concesión de permisos, dependiendo de las necesidades del proyecto. En este caso para estar seguros de que va a ejecutar de manera satisfactoria los test, se seleccionan todas las casillas.

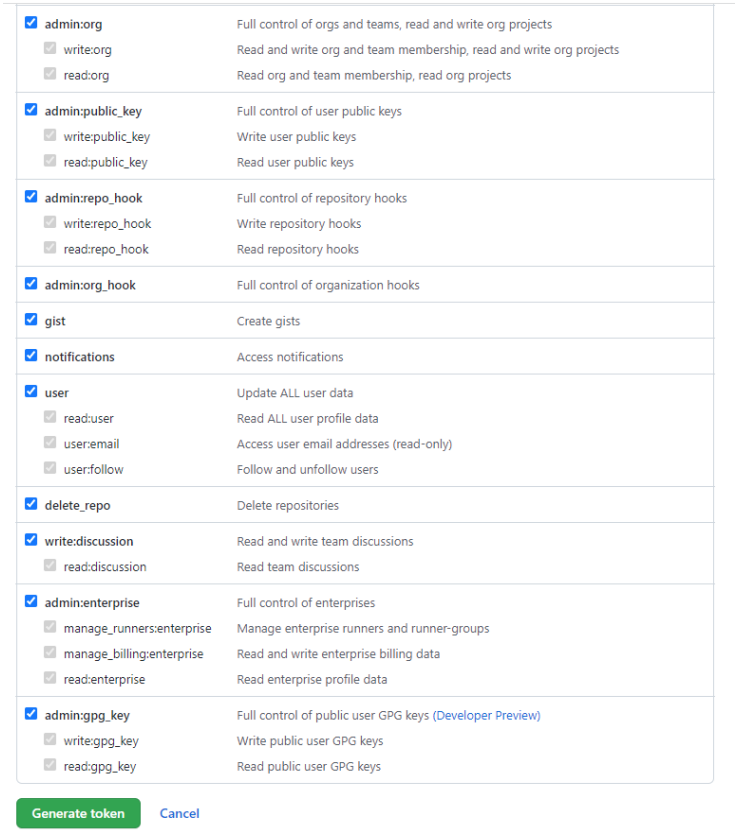


Figura 28 Permisos token Snyk

Marcados los permisos deseados, se pulsa el botón *generate token*. Una vez generado el *token*, se introduce en el apartado correspondiente para que pueda ser utilizado y acceder al APL de GitHub.

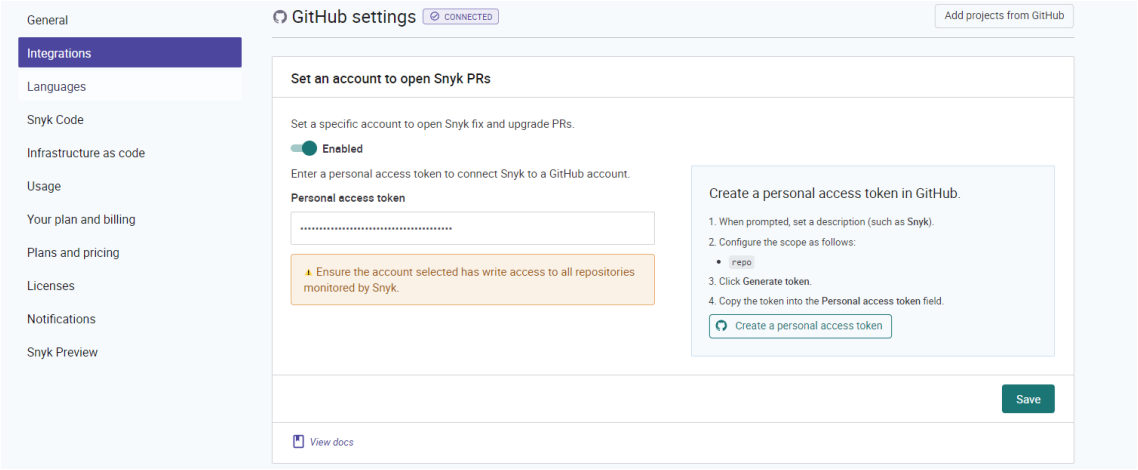


Figura 29 GitHub settings Snyk

Se hace uso de la aplicación de escritorio de GitHub [87], y de un *idle* para facilitar la edición del código del proyecto, en este caso se utiliza el Visual Studio Code [88]. Se podrá trabajar de manera paralela con el proyecto en local (almacenado en el equipo de trabajo) y el repositorio web de GitHub. En la siguiente imagen, se indican ambas rutas.

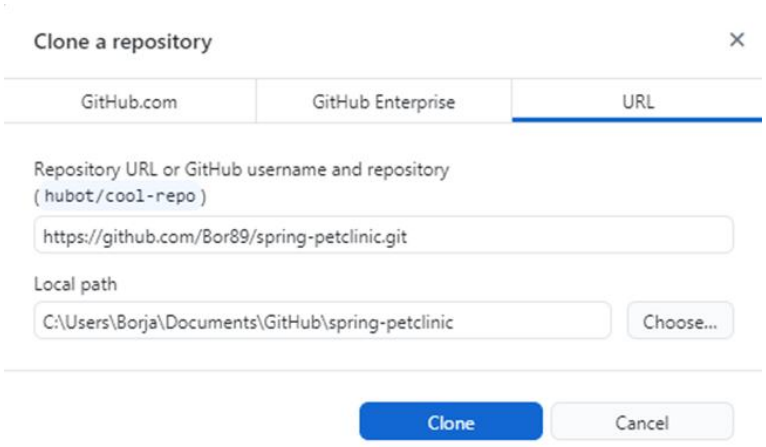


Figura 30 Rutas repositorio GitHub Desktop

Con el idle (*Visual Studio Code*), se abre todo el proyecto y se añade una dependencia al código para simular que este contiene la librería con la vulnerabilidad log4j, dentro del fichero pom.xml.

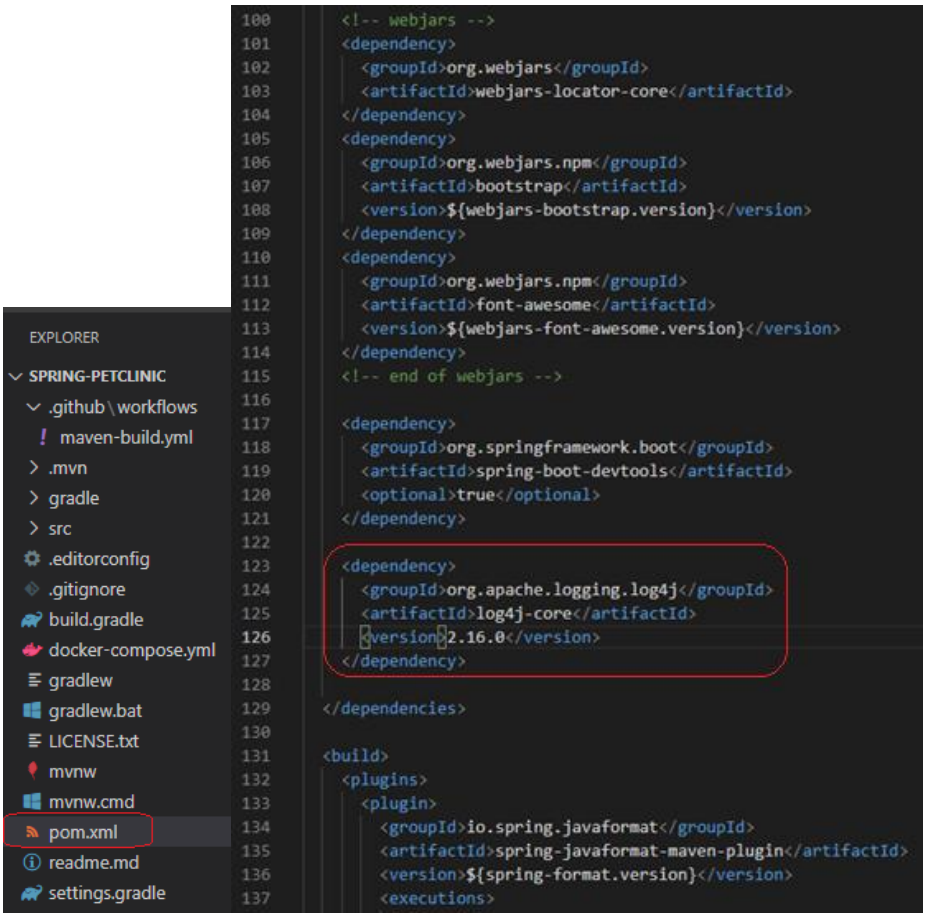


Figura 31 Inserción dependencia log4j

Se edita el número de la versión para asegurar que es la versión inicial comprometida con la que saltaron las alarmas (2.14.1).

```
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
  <version>2.14.1</version>
</dependency>
```

Figura 32 Inserción dependencia log4j V14

Desde GitHub *Desktop*, En el *current branch* que se está trabajando (log4j), se puede observar cómo detecta los cambios ejecutados.

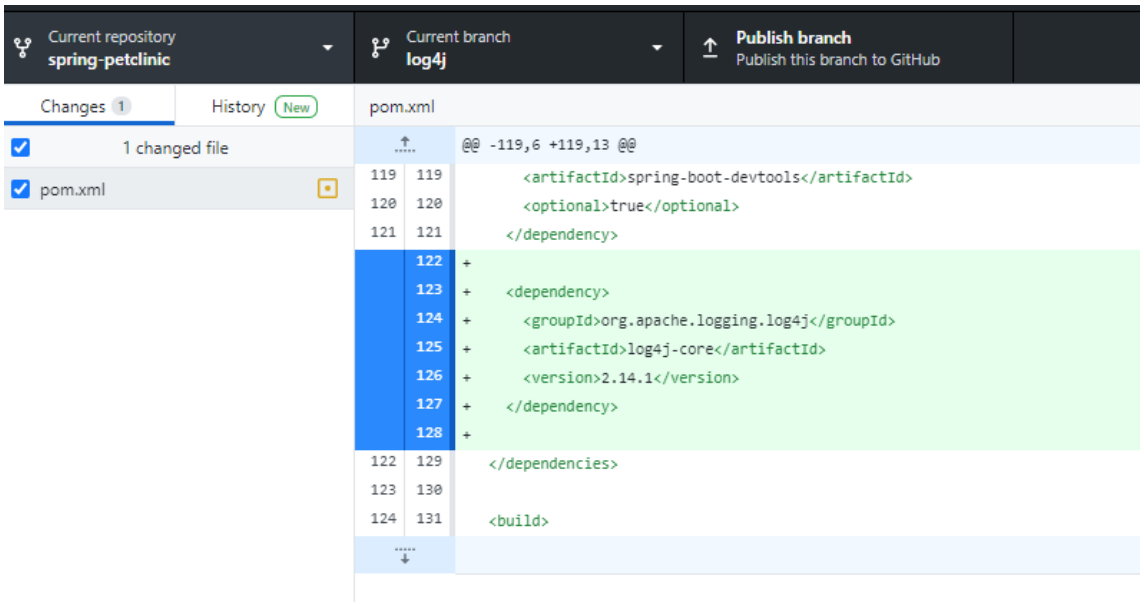


Figura 33 Cambios código en GitHub Desktop

Ahora en GitHub web, se pulsa el botón *create pull request*, para confirmar la solicitud de extracción del proyecto. Se puede observar que se genera un error.

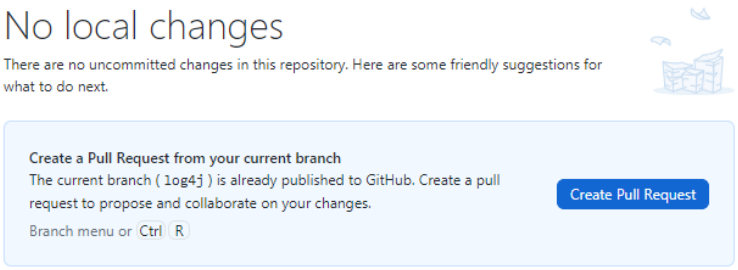


Figura 34 GitHub Pull Request

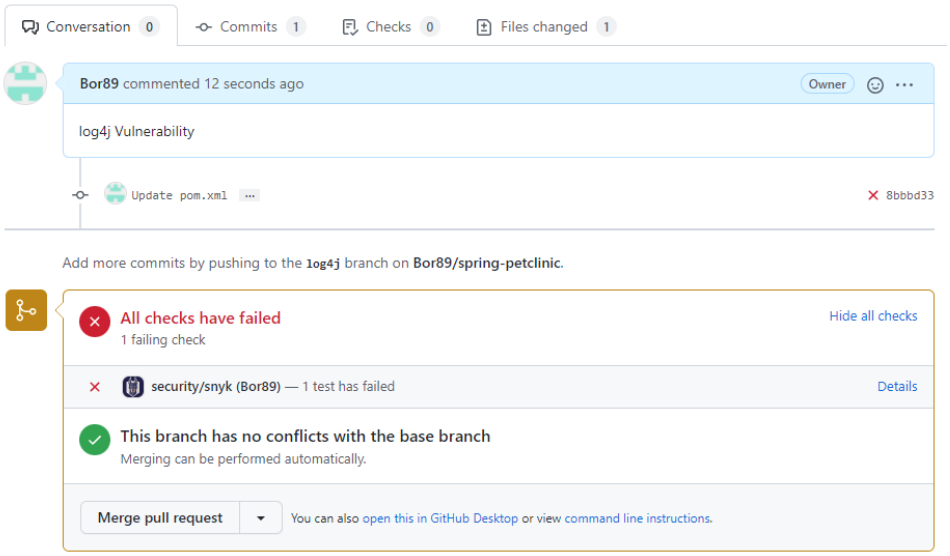


Figura 35 Branch error Snyk

Si se pulsa en "*Details*" y a continuación en "*View test page*".

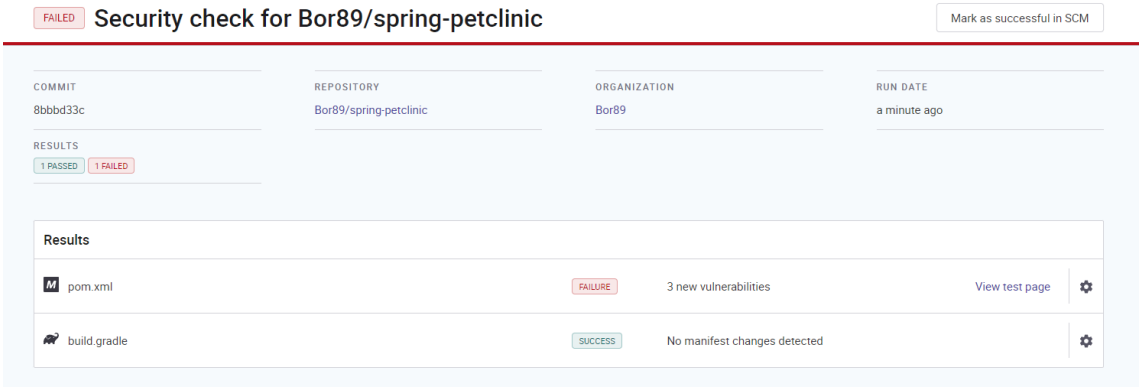


Figura 36 Security check

Se puede observar que Snyk ha detectado la vulnerabilidad log4j que afecta a Apache, la cual, ha sido introducida de manera intencionada en el proyecto.

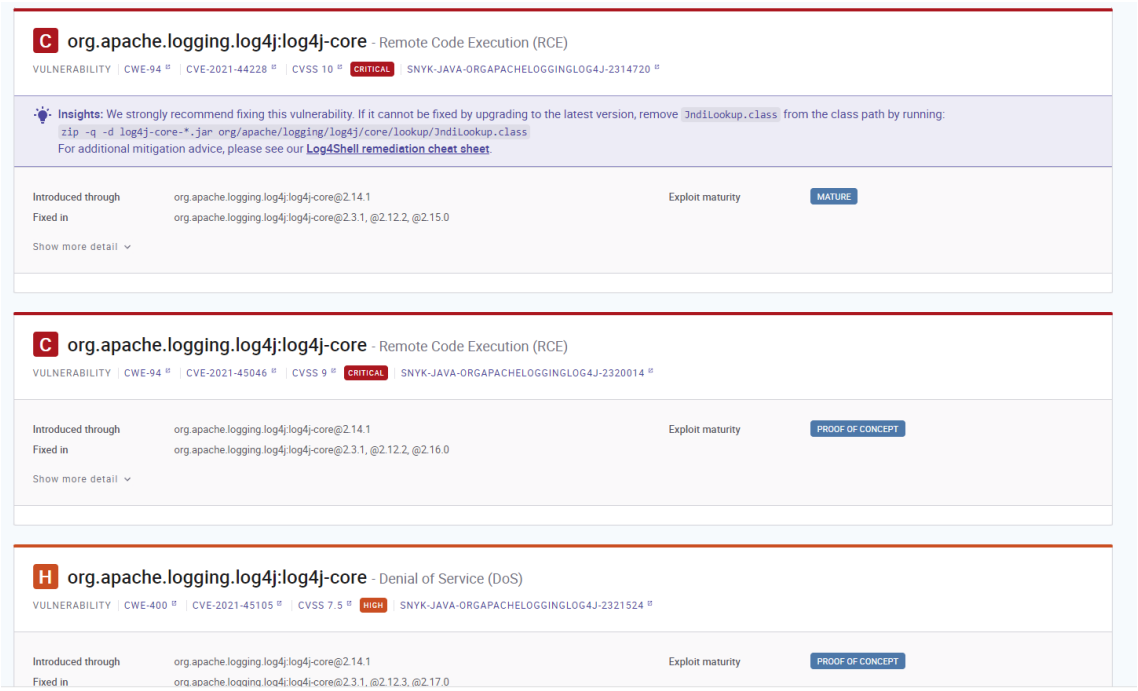


Figura 37 Detección vulnerabilidad Snyk

Integración de SonarCloud con GitHub

Los pasos son muy parecidos a los que se han seguido con Snyk. Se debe acceder a la web [89] y realizar el asistente para dar autoridad a SonarCloud para que acceda a la información de GitHub y después indicar los proyectos específicos.

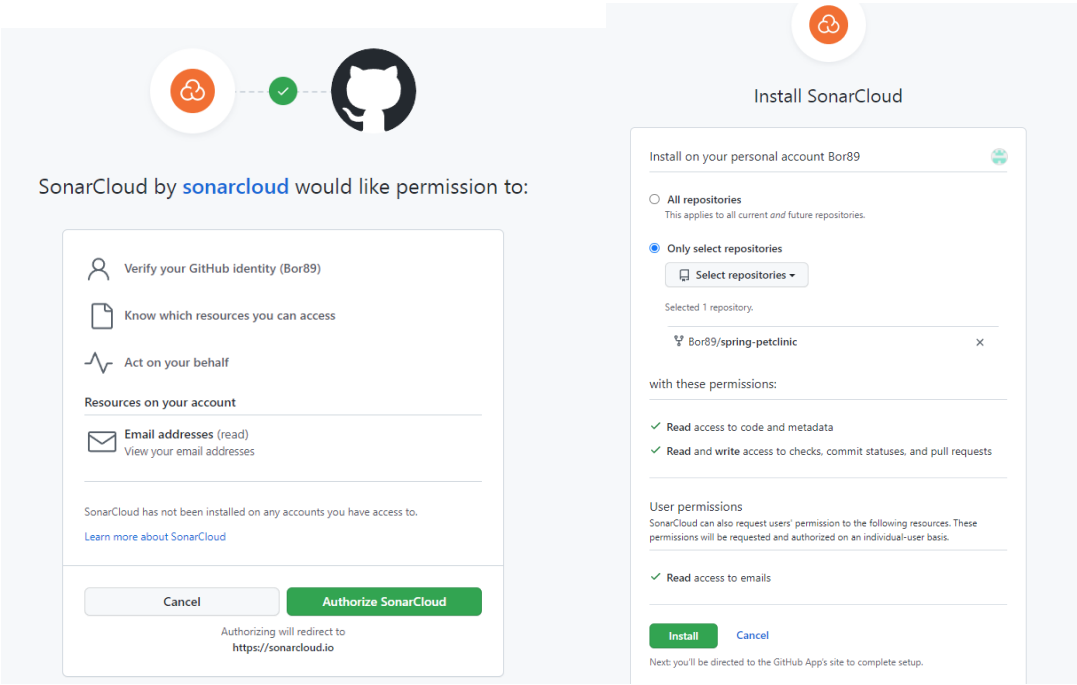


Figura 38 Autorización SonarCloud

Ahora se debe indicar el método de análisis deseado que llevará a cabo SonarCloud, en este caso, se selecciona GitHub.

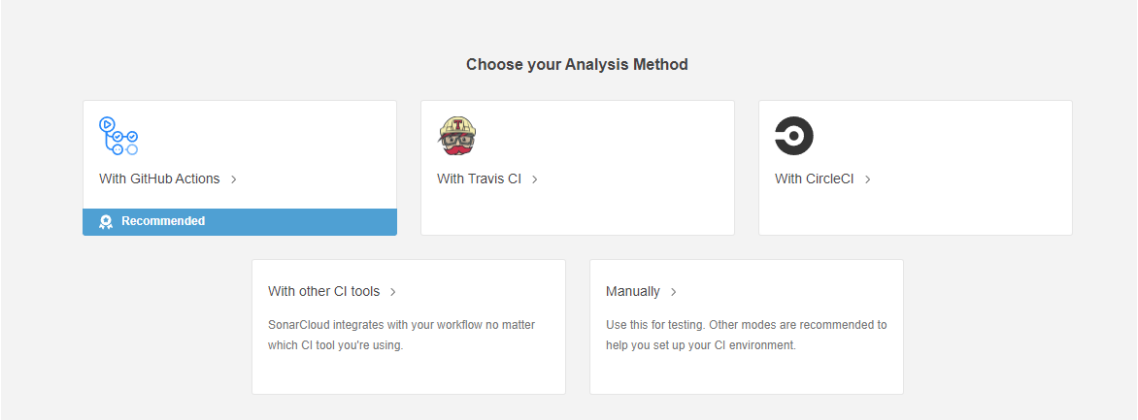


Figura 39 Método análisis SonarCloud

Para realizar los análisis del código de GitHub, SonarCloud solicita realizar dos acciones. La primera generar un *token* para utilizar como secreto durante las conexiones seguras entre ambas plataformas.

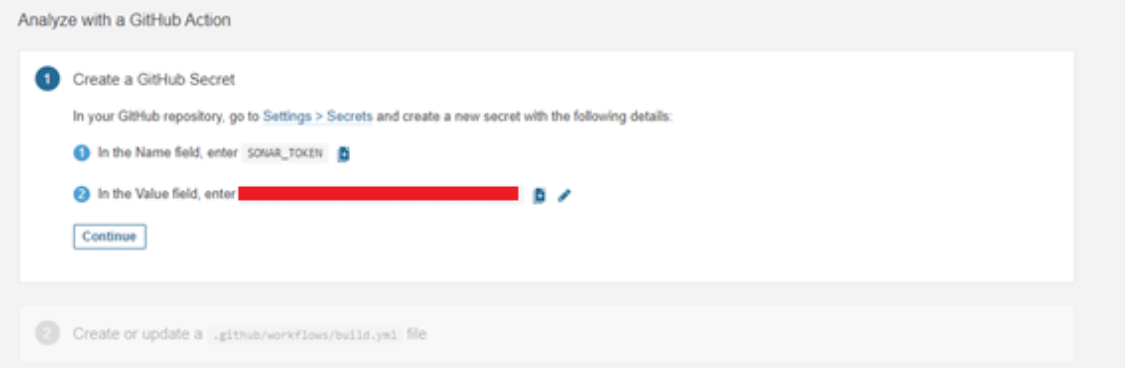


Figura 40 Secreto entre SonarCloud y GitHub

La segunda es elegir el tipo de *build* (proceso de compilación y ejecución), en este caso Maven [90].

Figura 41 Selección build Maven

Además, se deben añadir un par de líneas de código como dependencias web en el fichero pom.xml, para que se realice la conexión con la plataforma de SonarCloud. Y crear el fichero build.yml para que cuelgue de github\workflows y a continuación, copiar y pegar en este nuevo fichero el código que proporciona la web de SonarCloud.

```
<properties>

  <!-- Generic properties -->
  <java.version>1.8</java.version>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

  <!-- Web dependencies -->
  <webjars-bootstrap.version>5.1.3</webjars-bootstrap.version>
  <webjars-font-awesome.version>4.7.0</webjars-font-awesome.version>

  <jacoco.version>0.8.5</jacoco.version>
  <node.version>v8.11.1</node.version>
  <nohttp-checkstyle.version>0.0.4.RELEASE</nohttp-checkstyle.version>
  <spring-format.version>0.0.27</spring-format.version>

  <sonar.organization>bor89</sonar.organization>
  <sonar.host.url>https://sonarcloud.io</sonar.host.url>

</properties>
```

Figura 42 Líneas código conexión SonarCloud

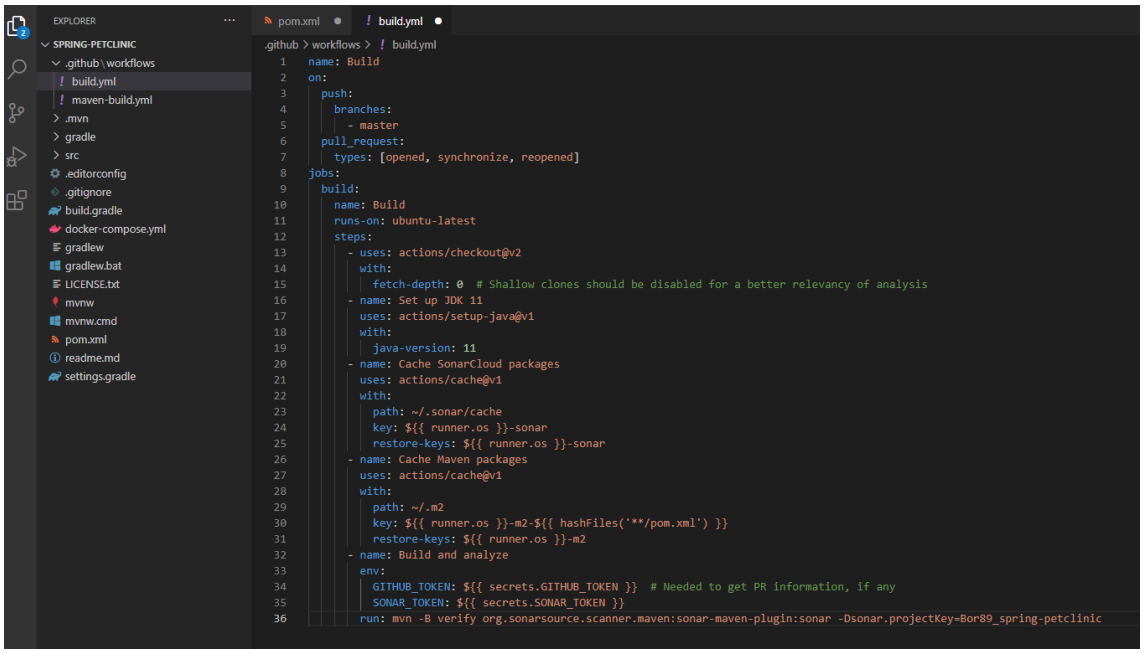


Figura 43 Build.yml SonarCloud

Con estos cambios se consiguen los comandos Maven necesarios para construir el artefacto de la aplicación y sea capaz de intercambiar información con SonarCloud.

Ahora, en la web de GitHub, accediendo al menú de Actions\Workflows, se puede observar que están corriendo diferentes tareas sobre el proyecto.

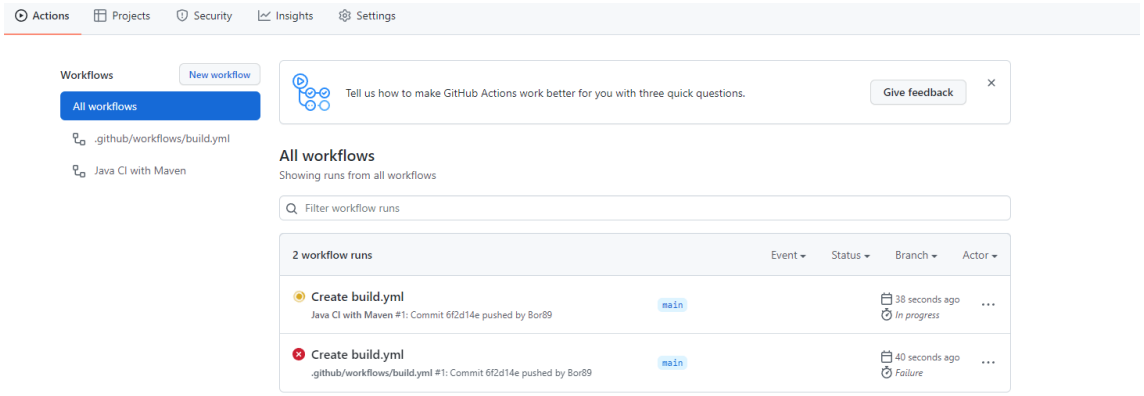


Figura 44 Tareas 1

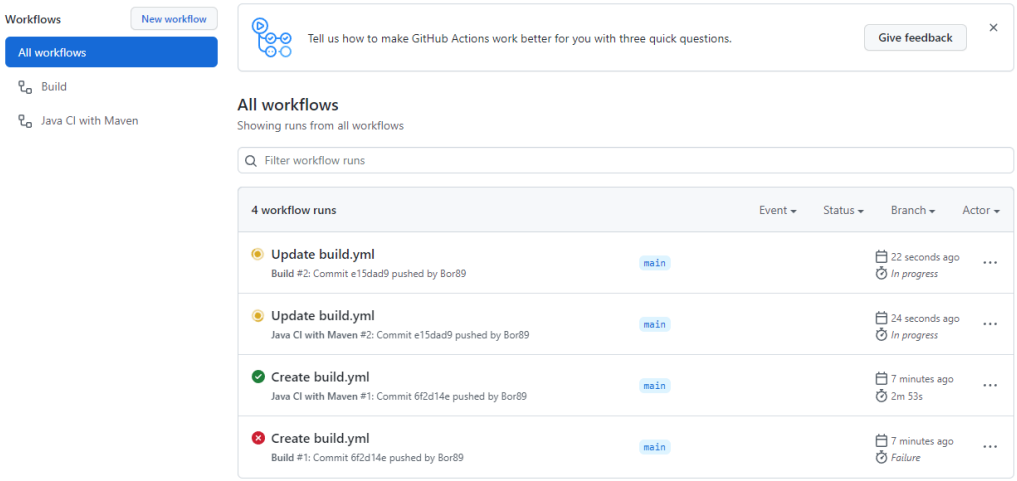


Figura 45 Tareas 2

Si se hace clic por ejemplo, en "*Update build.yml*", se puede ver el log que muestra las tareas que está realizando en tiempo de ejecución.

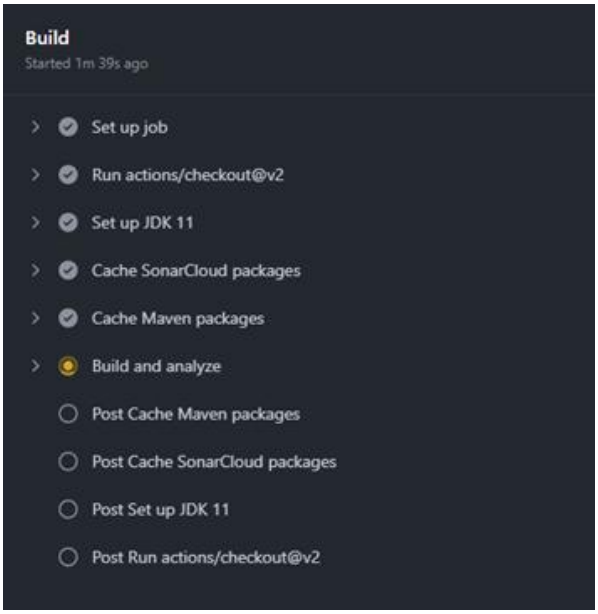


Figura 46 Build and analyze

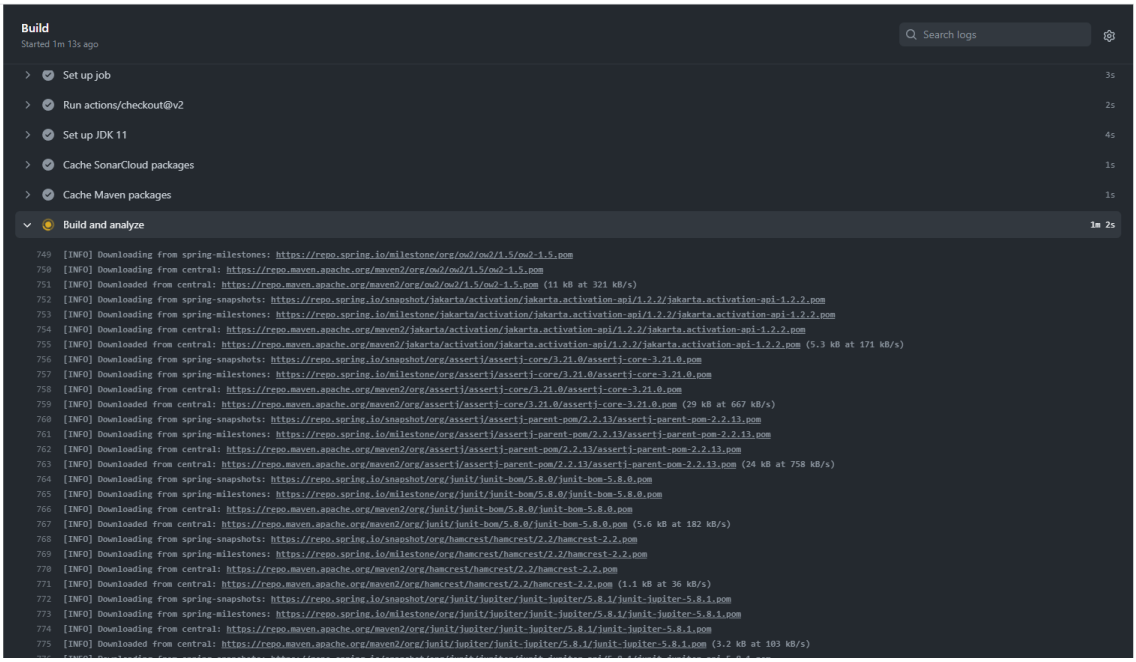


Figura 47 log tareas

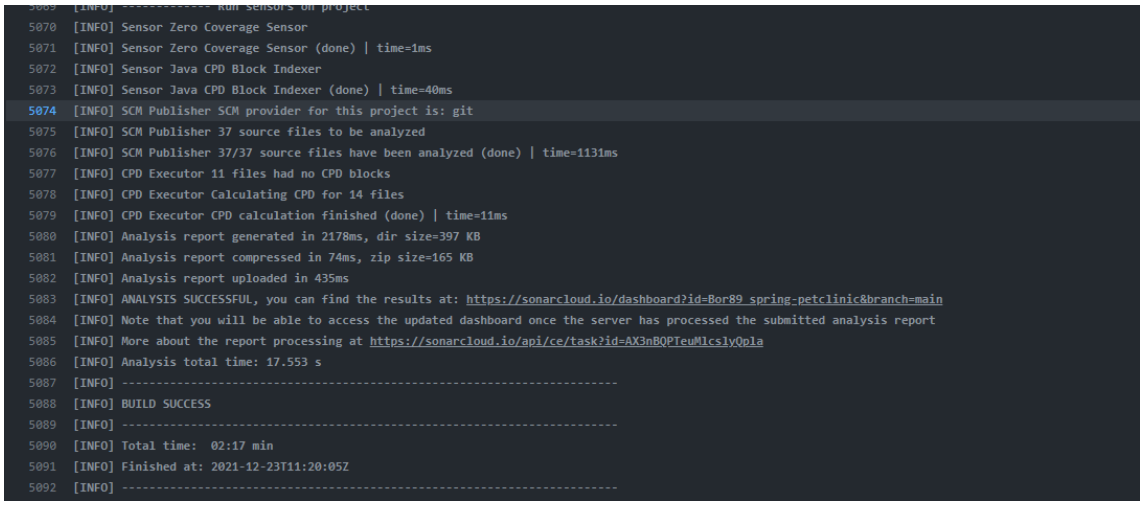


Figura 48 log tareas 2

En la web de SonarCloud se observa que se han detectado 9 vulnerabilidades.

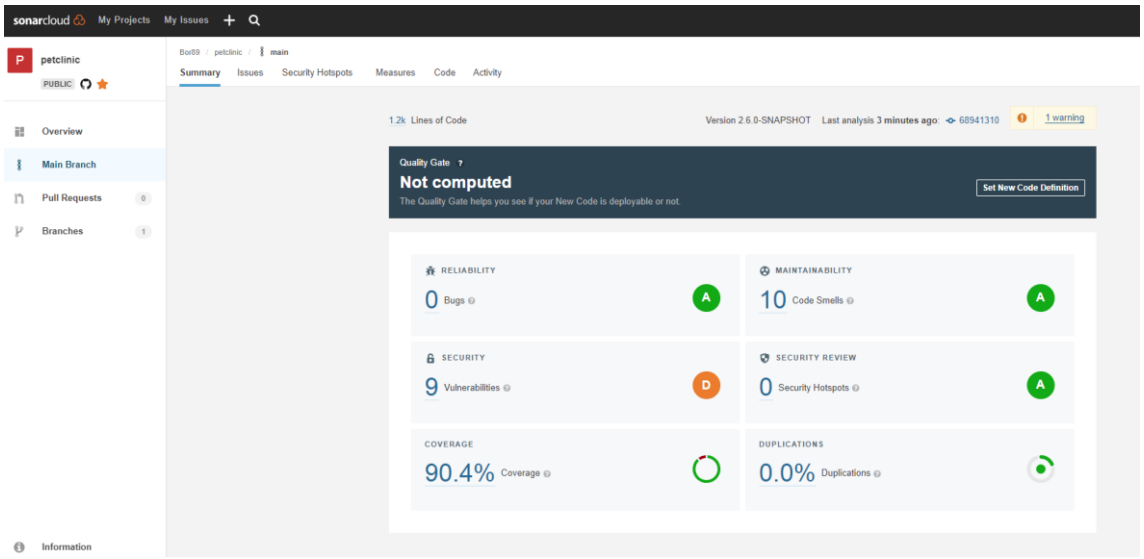


Figura 49 Vulnerabilidades SonarCloud

Finalmente, desde la web de Github, se pueden ver todos los test que se han configurado y que en este momento, se encuentran analizando el código del proyecto.

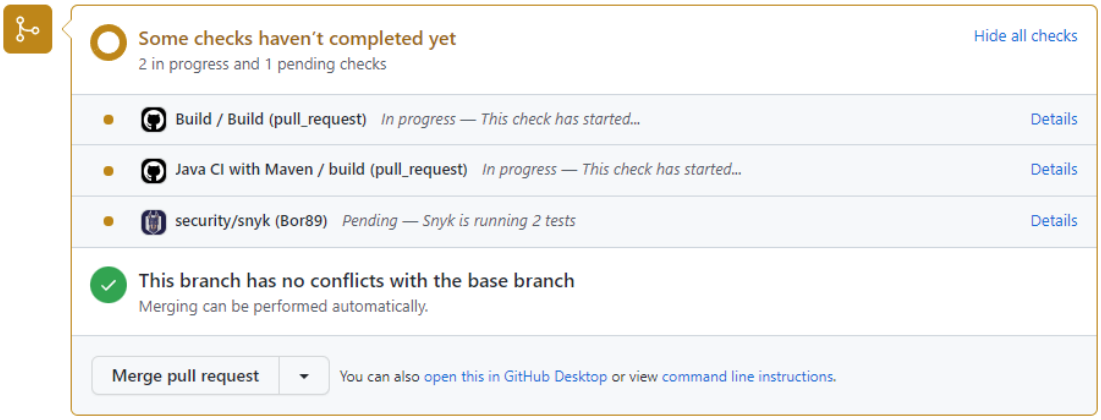


Figura 50 Test GitHub

Donde Snyk se ocupa de las vulnerabilidades de seguridad y SonarCloud de los problemas en la escritura del código.

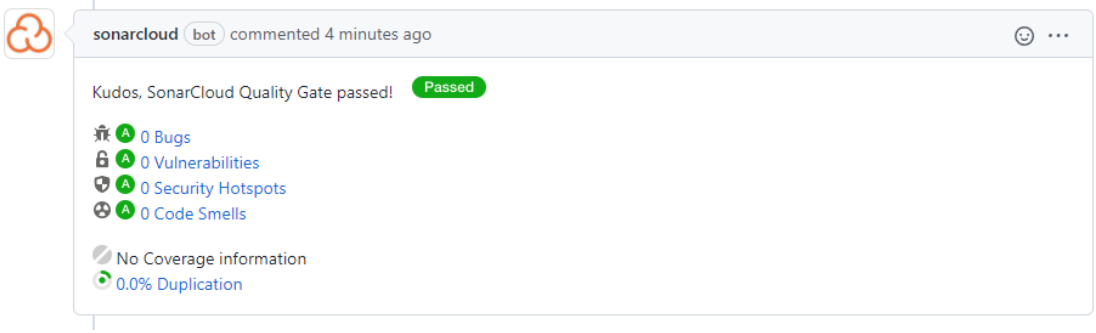


Figura 51 SonarCloud Quality gate