SecDevOps: Análisis de contenedores Docker e integración de herramientas SAST y DAST

Michelle López Rodríguez

11/02/2020

Resumen– La forma en la que creamos y desarrollamos código, cada vez requiere una respuesta más rápida a las necesidades de un servicio o de un cliente y a la vez que sea susceptible a los cambios para lograr una estabilidad a nivel funcional. Pero muchas veces lograr esa estabilidad no se consigue una vez finalizado el ciclo de desarrollo, sino que necesita un mantenimiento y una monitorización una vez está en producción. DevOps pretende dar una repuesta a este problema creando o generando un flujo de trabajo entre el equipo de Desarrollo y el de Operaciones, De este modo consiguen involucrarse desde la fase inicial de requerimientos hasta la fase final de feedback donde obtenemos un flujo continuo de comunicación sobre los productos o sistemas desarrollados ya sea mediante encuestas, chats, redes sociales o las herramientas de ticketing por parte de clientes finales. La seguridad es un punto de inflexión en la mayoría de desarrollos incluido DevOps por este motivo nace SecDevOps para integrar un nuevo equipo al flujo de trabajo añadiendo la seguridad desde la fase inicial. La capacidad de autoaprendizaje es muy importante en un modelo como este ya que la mejora debe ser continua sí se busca una estabilidad.

Palabras clave– DevOps, metodologías ágiles, integración continua, desarrollo, contenedores, funcional, monitorización, mantenimiento, feedback, estabilidad, ticketing.

Abstract– The way in which we create and develop code, increasingly requires a faster response to the needs of a service or a customer and at the same time is susceptible to changes to achieve stability at the functional level. But often achieving this stability is not achieved after the development cycle, but requires maintenance and monitoring once it is in production. Devops intends to provide a response to this problem by creating or generating a workflow between the Development and Operations teams, In this way they get involved from the initial phase of requirements to the final phase of feedback where we get a continuous flow of communication about products or systems developed either through surveys, chats, social media or ticketing tools by end customers. Safety is a turning point in most developments including Devops for this reason SecDevOps is born to integrate a new equipment to the workflow adding safety from the initial phase,the capacity of self-learning is very important in a model like this since the improvement must be continuous if a stability is sought.

Keywords– DevOps, agile methodologies, continuous integration, development, containers, functional, monitoring, maintenance, feedback, stability, ticketing.

1 Introduction

En el momento tecnológico en el cual vivimos, las metodologías ágiles [1] están en pleno apogeo a causa de los constantes cambios que exige la industria que afectan de manera directa a la calidad del software, plazos de entrega, al presupuesto y por último a la forma de gestión de los mismos. Antes de vivir este cambio

• E-mail de contacto: michelle.lopezr@e-campus.uab.cat
• Mención realizada: Tecnologías de la Información
• Trabajo tutorizado por: Àngel Elbaz (deic)
• Curso 2019/2020

tecnológico, la mayoría de desarrolladores utilizaban métodos o modelos más estáticos como Waterfall donde cada fase se ejecuta de manera secuencial. En algunos casos impide realizar cambios a mitad del desarrollo para añadir nuevas funcionalidades o requerimientos al proyecto. Este tipo de implementaciones en muchos casos no se pueden adaptar a un desarrollo ágil aunque la forma de trabajo es más segura. Con el paso del tiempo estos sistemas se volvieron más complejos para satisfacer la demanda de necesidades y los diferentes problemas que surgían a medida que avanzaba la tecnología.

Estas metodologías han experimentado una gran evolución a lo largo de este tiempo, Ahora los desarrollos en un gran alto porcentaje utilizan modelos ágiles que permiten la integración y correción de cambios en las diferentes iteraciones o fases del proyecto para de esta manera conseguir subir a producción una release de una manera continua. DevOps actúa más como un complemento al proceso de desarrollo ágil ya que permite facilitar los procesos de integración y entrega continúa asegurando que todas las versiones de código generadas están listas para poner en producción. Estas nuevas entregas se pueden realizar de forma regular (semanalmente, diariamente o varias veces al día) de esta manera se consigue el paso a producción un código más seguro, estable y eficiente.

¿Pero, donde nos dejamos la seguridad?, Como ya sabemos la seguridad es un tema muy importante en todos los ámbitos, Más aún en el corporativo. Esto no siempre es así en los modelos de desarrollo ya que en muchos casos esto cobraba menor importancia o se planteaba al final del proyecto. Necesitamos garantizar que la seguridad no sea un departamento aislado al que solo se acuda una vez finalizado el proyecto y se ocupen de hacer que nuestro sistema sea seguro. Si no todo lo contrario que se integren en todas las fases del proyecto. Con los métodos ágiles de desarrollo esto se complica aún más ya que cada sprint o iteración se genera un versión de código nueva lo que generaba un conflicto entre ambos equipos. Con **SecDevOps** el objetivo es integrar el equipo de seguridad en todas las etapas del proyecto. Esto nos ayudara a incluir procesos de seguridad desde el análisis de requerimientos y en algunos casos tengamos que crear sistemas de seguridad. Se podrían ejecutar pruebas de seguridad en todas las fases y contra el código. Otro punto importante es la automatización de esas comprobaciones. Estos análisis pueden ser de dos tipos: En primer lugar las pruebas DAST, actúan contra el propio software o aplicación atacándose como un hacker en busca de vulnerabilidades. Por el contrario las pruebas SAST, examinan el código fuente propio para encontrar posibles fallos en la forma de programar. Estos y otros conceptos son los que abordaremos y analizaremos en este trabajo.

2 Objetivos

Uno de los objetivos globales de este Estudio es la seguridad, como ya sabemos es uno de los puntos más importantes en cualquier desarrollo o proyecto y cada vez exige ser incluida desde el inicio del mismo. De esta manera se implementa la seguridad desde el ciclo de requisitos o definición de proyecto.

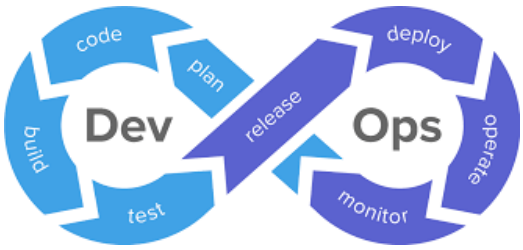

Fig. 1: Principales logos de las Herramientas a Estudiar

2.1 Objetivos Principales

- Analizaremos el modelo SecDevOps, la implantación de procedimientos de automatización y análisis de código estableciendo un ciclo de vida de desarrollo de software seguro.
- Determinar pruebas automáticas estáticas y dinámicas de seguridad mediante integración continua.
- Analizar la seguridad de los contenedores Docker y el impacto en DevOps.
- Estudiar el impacto de las metodologías ágiles y DevOps el impacto en el ámbito de las TI.
- Análisis de la situación actual de DevOps.
- Plantear un ciclo de vida de desarrollo de software seguro.
- Definir las herramientas DevOps a utilizar en el estudio.
- Investigar y plantear pruebas de seguridad tanto estáticas como automáticas.

2.2 Objetivos Secundarios

- Estudiar las motivaciones por las que una empresa adoptan o no DevOps.
- Profundizar en el uso de herramientas como Jenkins y Docker.
- Se puede implementar DevOps en un modelo Mainframe.

3 Estado del Arte

Las metodologías han experimentado una gran evolución a lo largo de este tiempo, antes se utilizaban modelos duros o más tradicionales imponiendo una disciplina de trabajo en el desarrollo de software con el objetivo de conseguir un resultado más eficiente. Se centraban en la gestión de control de los procesos, definición de roles, escoger herramientas y una buena documentación. Pero estas no lograban adaptarse a los posibles cambios que pudieran surgir durante el proceso de desarrollo de un producto o software, las metodologías ágiles surgen como respuestas a estas necesidades. Es importante tener el conocimiento de ambas para poder escoger las que más se adapte al tipo de proyecto que desarrollamos.

Las llamadas metodologías ágiles, se corresponde a un grupo de metodologías aplicadas al ciclo iterativo en la creación y desarrollo del software fomentando la colaboración entre diferentes equipos. El objetivo principal según Wikiversidad Contibutors[1] es agilizar el desarrollo y la entrega del producto en cada iteración antes que la elaboración de la documentación o en el estricto cumplimiento de los protocolos. Las metodologías ágiles más utilizadas en el desarrollo y gestión de software son **Scrum, Kanban, Lean Software Development, XP** (eXtreme Programming), pero existen otras metodologías recogidas en Agile Alliance [2].

3.1 Kanban

El método Kanban es de origen japonés que significa "Tarjeta Visual" es un medio para gestionar, diseñar y mejorar los sistemas de trabajo. Funciona como un sistema de señalización para comunicar información relativa a la ejecución del trabajo y como herramienta de gestión y control del proyecto. Los Beneficios claves del Método es que ayuda a la estimulación del rendimiento permite detectar posibles problemas y ajustar el flujo para ganar eficiencia. Además favorece la organización y colaboración a través del enfoque visual entre el equipo, actualmente se utilizan herramientas web para la compartición de este tablero y poder acceder desde cualquier sitio. Otra ventaja es la posibilidad de la distribución del trabajo ya que la visualización general de los trabajos en curso y menos tiempo dedicado a la distribución y presentación de los trabajos evitando la formación de cuellos de botella.

3.2 Scrum

Es una metodología que se basa en un modelo de desarrollo iterativo e incremental, enfocado más en las personas que en los procesos. Las iteraciones se denominan Sprints y cada una de ellas finalizan con la entrega de una versión operativa del producto. Esto genera un incremento continuo del producto el ciclo se repite hasta completar el producto y el cliente lo verifique. Scrum gestiona el progreso del trabajo a través de reuniones breves diarias donde el equipo revisa el trabajo realizado del día anterior y el previsto para el siguiente. Los Sprints són la base de este modelo.

La mayoría de metodologías ágiles comparten varias características, buscan la comunicación entre equipos, los requerimientos del cliente y no solo están definidas al desarrollo del software sino que se pueden adaptar al cualquier tipo de proyecto. Se recomienda que las empresas adopten estos métodos para eliminar emplear esfuerzos sin planificación, reuniones inútiles que no generan productividad entre otros.

3.3 DevOps

Con la llegada de las metodologías ágiles, se convirtieron en la herramienta ideal para recuperar la confianza en el desarrollo del software, pero se olvidaron de lo que pasa luego que el software está en producción como mantenerlo o de qué manera darle soporte en caso de incidencias. La parte de operaciones quedaron en un segundo plano, **DevOps** pretende aunar ambos equipos para que la confianza lograda con las metodologías ágiles se extienda en todo el ámbito TI.

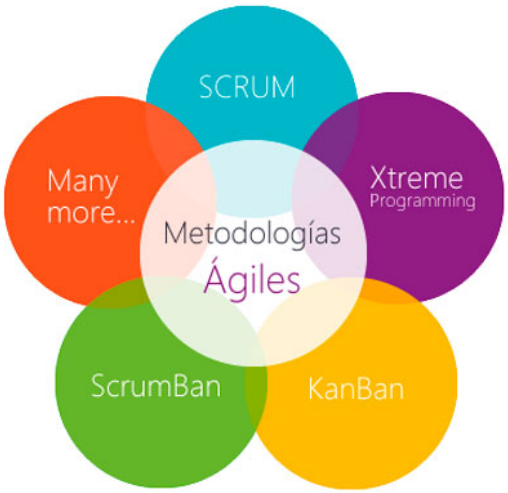

Fig. 2: Metodologías ágiles más utilizadas

Actúa más como un complemento al proceso de desarrollo ágil ya que permite facilitar los procesos de integración y entrega continúa asegurando que todas las versiones de código generadas están listas para poner en producción. De esta manera DevOps genera un nuevo flujo de trabajo entre ambos equipos que facilitan la implantación y despliegue de código nuevo. Estas nuevas entregas se pueden realizar de forma regular (semanalmente, diariamente o varias veces al día) de esta manera se consigue el paso a producción un código más seguro, estable y eficiente.

Estas nuevas entregas se pueden realizar de forma regular (semanalmente, diariamente o varias veces al día) de esta manera se consigue el paso a producción un código más seguro, estable y eficiente.

¿Por qué elegir DevOps?

No existe un método que por sí solo pueda garantizar el éxito de nuestro proyecto, Sin embargo existen evidencias qué el uso de prácticas DevOps está compuesto por muchos factores como la mejora organizativa, se adapta a las condiciones del mercado, efectividad en las operaciones y un liderazgo en la dirección de proyectos. Varios estudios que hablan de la mejora del rendimiento demuestran que empresas del ámbito tecnológico que introducen prácticas DevOps pueden optar a objetivos de mayor alcance y mejoran notablemente sus resultados; Se presenta como una nueva cultura organizativa que ayuda en el trabajo diario, mejora la comunicación y operatividad entre equipos, se documentan mejor los procesos, permite aprender de los errores, hacer cambios y generar nuevas ideas. La forma de medir esta mejora de rendimiento es mediante la frecuencia de despliegue y tiempo de espera para ejecutar cambios, por otra parte la estabilidad se mide en el tiempo de solución de errores o en el tiempo de recuperación. Para mejorar estos resultados es necesario emplear prácticas que mejores estas medidas de rendimiento y estabilidad. La mejora continua implica que el software debe cumplir con todos los requisitos para estar en producción, para ello es necesario la solidez del entorno ya que aplicaremos repetidamente cambio y necesitamos un entorno fiable. Las empresas que tienen la seguridad integrada a lo largo del ciclo de vida de entrega de software son más propensas a

utilizar prácticas **DevOps** [9].

3.4 Entrega e Integración Continua (CI/CD)

Para mantener la competitividad de una empresa, es necesario ofrecer aplicaciones de calidad que se adapten a las necesidades, para ello muchas empresas han incluido el concepto de entrega continua **(Continuos Delivery)** este modelo permite agilizar el proceso de entrega de software. La entrega continua ayuda a mejorar la eficiencia ya que durante el desarrollo se mantiene siempre en un estado de entrega, requiere la colaboración eficaz de todos los equipos. El otro concepto que hablaremos es la Integración continua **(Continuos Integration)**, como ya sabemos es habitual que dentro de los equipos existan grupos de desarrolladores encargados de diferentes partes del código o que estén desarrollando una funcionalidad que se divide en varias. Pues una forma de unificar todo el trabajo realizado por diferentes grupos o personas es a través de una rama máster donde fusionamos todo el código generado. Cada vez que se aplica un cambio en la rama master, se ejecuta una batería de pruebas para detectar posibles errores y en algunos casos evitar errores funcionales. Realmente este proceso es necesario para alcanzar la entrega continua, el objetivo principal es que el proceso de aplicar cambios sea un proceso sencillo y rutinario. Esto permite ahorrar tiempo y que este pueda ser utilizado en otras tareas que lo requieran añadiendo valor a la empresa. Según Martin Fowler [3] para una buena integración continua debemos considerar los siguientes principios:

- Mantener un solo repositorio de código.
- Automatizar la construcción.
- Automatizar el test de ensamblado.
- Todos los miembros del equipo suben los cambios al repositorio central.
- Cada commit debe integrar los cambios al repositorio central.
- Fijar tiempos cortos de ensamblado.
- Hacer pruebas en un entorno BETA.
- Mantener una construcción rápida.
- Fácil de obtener las últimas entregas.
- Todos los miembros del equipo pueden ver lo que está pasando.
- Automatizar Implementación.

Después de todo lo estudiado con anterioridad, podemos sostener que DevOps no es más que otra forma de mejorar las prácticas de trabajo actuales. Sabemos también que las metodologías tradicionales en algunos aspectos ya no se adaptan a los continuos cambios que exige el mercado y la competitividad atrás quedan los modelos secuenciales que se estimaban en meses o años, aunque existen sectores que los siguen desarrollando debido a que las funcionalidades dudosamente cambian o descubrieron que se adecuán mejor a las actividades que producen. En todo caso se han

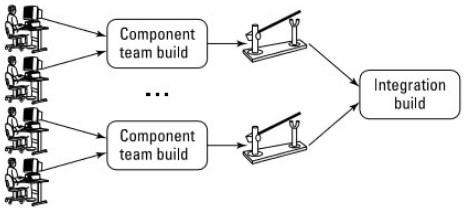

Fig. 3: Colaboración a través de la integración continua (DevOps For Dummies, 2017).

visto desplazadas por modelos más ágiles, flexibles a los cambios y entrega rápida que pueden llegar a ser 300 despliegues por día dato que proporciono Amazon en 2011 en una conferencia [4].

Actualmente es difícil encontrar personal que no haya trabajado o este familiarizado con el marco ágil. DevOps es posible aplicarlo en aquellos lugares donde exista la posibilidad de mejora, Un estudio [5] argumenta que la incorporación de esta práctica les ha permitido implantar código hasta 30 veces con más frecuencia con respecto a su competencia. Y en otras publicaciones aseguran que menos del 50% las implantaciones llegan a fallar en producción.

Entonces porque solo un tercio de las empresas han incluido DevOps en algunas de sus prácticas, posiblemente se deba a muchos factores como el tiempo de implantación o la resistencia a los cambios. ¿Por qué cambiar lo que ya funciona?, ¿Cuánto tiempo me va a tomar cambiar el método de trabajo?, ¿Mis equipos se podrán adaptar?, ¿Necesitamos formación nuevas Habilidades o Conocimientos? etc...

Algunos sectores no están dispuesto a parar sus operaciones y se resisten al cambio, también se enfrenta a los posibles problemas durante el tiempo de implantación y lo que tardaran en adaptarse a los nuevos procesos. Por eso existen aún tecnologías como mainframe, desarrolladores de cobol y se sigue desarrollando (aunque en un porcentaje menor) con modelos **Waterfall** tal vez por el gasto que implicaría un cambio o porque simplemente funcionen y sea estable.

El movimiento DevOps no solo se centra en el desarrollo sino que le interesa el mantenimiento del mismo y que esté preparado para someterse a una mejora en cualquier momento también la integración de equipos de Testing y los que proporcionan el soporte y mantenimiento de los entornos de producción. El estudio patrocinado por **CA Technologies denominado "Accelerating Velocity and Customer Value with Agile and DevOps**" [5] asegura que encuesto a 1.770 ejecutivo del sector TI tanto de Europa, Oriente Medio y África sobre el uso de modelos ágiles y DevOps y que impacto ha generado en sus actividades. Para medir este impacto es necesario nombrar una serie de indicadores que van desde el rendimiento, la productividad o la satisfacción del cliente lo veremos de una forma más clara en el siguiente gráfico:

Además el tiempo que tardan en desarrollar y lanzar una nueva versión o aplicación se ha reducido en el caso de las Ágiles de 10 semanas a 8 semanas y en el caso de **DevOps** de las 9 semanas a casi 7 semanas de media. El estudio también destaca el nivel de madurez que se encontraban los encuestados en el uso de las metodologías estaban divididos en 2 grupos de usuarios: Avanzados y Básicos o no Usuarios y el resultado es que las empresas españolas se encuentra

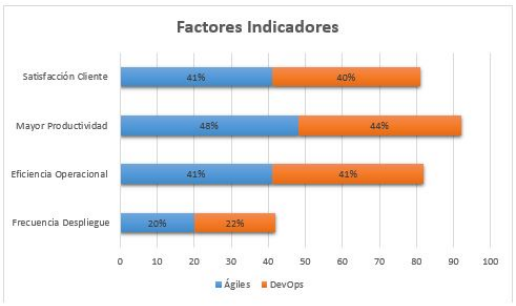

Fig. 4: Principales indicadores de impacto en el negocio (CA Technologies, 2016)

	FASE 1: Analisis de los Requerimientos y Estado del arte	12 días	lun 07/10/19	mar 22/10/19	
	Recopilación y análisis de información	6 días	lun 07/10/19	lun 14/10/19	
	Investigación del estado de arte	6 días	mar 15/10/19	mar 22/10/19	2
	FASE 2: Definición de la metodología y tecnologí	12 días	mié 06/11/19	jue 21/11/19	
	Especificación de las fases de software seguro	5 días	mié 06/11/19	mar 12/11/19	
	Aplicación y elección de técnicas en cada fase	7 días	mié 13/11/19	jue 21/11/19	5
	FASE 3: Creación del entorno de pruebas	7 días	jue 21/11/19	vie 29/11/19	
	Descargar el software elegido	2 días	jue 21/11/19	sáb 23/11/19	
	Instalación del Entorno Práctico	5 días	lun 25/11/19	vie 29/11/19	8
	FASE 4: Desarrollo e implementación del proyect	31 días	lun 09/12/19	vie 17/01/20	
	Escoger código para la demostración	7 días	lun 09/12/19	mar 17/12/19	
	Implementación del código en el entorno práctico	15 días	mié 18/12/19	mar 07/01/20	11
	Testeo de integración correcta	9 días	mié 08/01/20	vie 17/01/20	12
	FASE 5: Fase de test, pruebas y corrección de errores	6 días	sáb 18/01/20	vie 24/01/20	
	Comprobación de errores	3 días	sáb 18/01/20	mar 21/01/20	
	Corrección de código	3 días	mié 22/01/20	vie 24/01/20	15
	FASE 6: Verificación de resultados finales	3 días	sáb 25/01/20	mar 28/01/20	
	Validación de resultados	3 días	sáb 18/01/20	mar 21/01/20	
	Creación de demo	2 días	lun 27/01/20	mar 28/01/20	18
	FASE 7: Documentación Informe Final	3 días	mié 29/01/20	vie 31/01/20	
	FASE 8: Entrega memoria Final	7 días	vie 31/01/20	sáb 08/02/20	
	Creación de Paper TFG	7 días	sáb 01/02/20	sáb 08/02/20	

Fig. 5: Cronograma de Tareas y subtareas del proyecto.

entre las más avanzadas en el uso de metodologías ágiles cerca del 33% de las encuestadas y que empata con Francia y Alemania, Pero, al contrario en el uso de **DevOps** esta cifra cae al 27% y se queda detrás de países como Alemania (40%), Francia (39%) y Suecia (33%) entre los destacados.

Al menos en España aún hay mucho trabajo que hacer en este tema viendo las cifras podemos valorar que las metodologías tradicionales siguen jugando un papel importante en el sector de las TI frente al movimiento emergente de **DevOps** y del uso de modelos ágiles.

4 Metodología y Planificación

Utilizaremos la metodología Kanban [6] para el desarrollo de este TFG, este modelo nace en los años 40 y fue implementada por Toyota una producción bajo demanda llamada **"just in time"**. Los principios básicos de **Kanban** son: Visualizar el flujo de trabajo a través de 3 columnas por hacer, en proceso y hecho. Limitar el número de tareas que se ejecutan a la vez y por último gestionar el flujo de trabajo. Se establecerán iteraciones por semana, se limitara el número de tareas en proceso y una herramienta de gestión que nos permitirá visualizar el tablero. La aplicación **Trello** [7] es la herramienta de gestión que nos servirá para llevar un control sobre las distintas tareas del proyecto y su estado. Dicho aplicativo será compartido con el tutor del proyecto para que a su vez este informado todo el tiempo del trabajo realizado.

En el siguiente apartado se describirá las fases del proyecto y su duración.

4.1 Planificación

En la planificación inicial del TFG, se plantea las fases y tareas necesarias para llevar a cabo el proyecto. Durante estos meses se ha logrado completar la gran mayoría del proyecto incluyendo subtareas; están marcadas en color verde para poder diferenciarlas del resto.

Durante este tiempo han surgido retrasos y cambios en algunas de las tareas y fases. Se ha podido solucionar realizando un reorganización de las fechas de inicio y fin en los lugares afectados. También se ha llevado a cabo un enorme esfuerzo para cumplir con los plazos establecidos.

5 Resultados

5.1 Desarrollo

Como bien sabemos, existe una amplia gama de herramientas utilizadas y relacionadas con **DevOps**. La cual va creciendo cada día, por este motivo es inevitable tener muchas dudas a la hora de escoger la que mejor se adapte a nuestro proyecto.

En concreto en este han surgido muchas dudas y se han investigado las mismas para determinar sus beneficios y su viabilidad. Finalmente han sido escogidas las que mejor se adaptaban a los recursos que disponíamos en el momento de realizar el estudio.

Después de una pequeña investigación, decido crear un entorno virtualizado dentro de mi máquina física. Para elegir las herramientas de virtualización, como el sistema operativo y especificaciones necesarias para la instalación de mi entorno virtual.

A continuación describiremos todas las herramientas que utilizaremos en el entorno virtual:

Requisitos

La parte práctica de este proyecto se realizará en su mayoría utilizando software open source.

PC	
Equipo	**Hardware**
Dell Latitude E5470	i5 6300U 8GB SDRAM 250 GB

TAULA 1: TABLA HARDWARE

Herramientas	
SO/Aplicación	**Versión**
VMware Workstation Pro	15.5.1
Ubuntu Desktop	18.04

TAULA 2: TABLA SOFTWARE

5.1.1 Herramientas DevOps

Volviendo a **DevOps**, existe una gama amplia de herramientas que se pueden utilizar para emplear una solución, pero las escogidas en este proyecto son las siguientes:

Herramientas	Versión
Jenkins	2.2
Docker	CE
Github	App Web

TAULA 3: HERRAMIENTAS UTILIZADAS

5.1.2 Automatización e Integración/Entrega Continua (IC/DC)

Uno de los pilares fundamentales de DevOps es la automatización y la ejecución continua de versiones de código, así como también la capacidad de detectar posibles errores mediante el análisis con herramientas **SAST** Y **DAST**. Jenkins es la elegida para esta tarea, se ha convertido es una de las principales y más utilizadas herramientas de orquestación para este tipo de desarrollo. La nueva versión incluye **Jenkins Pipeline** [8] el cual permite incluir un set de plugins predefinidos para facilitar el despliegue desde el repositorio a los diferentes sistemas y que puedan ser utilizados por los usuarios de esta manera generamos la entrega continua.

Fig. 6: Logo de Jenkins.

5.1.3 Plugins SAST y DAST

Uno de los puntos importantes del modelo DevOps es el análisis del código antes, durante y después de la integración hasta llegar a una reléase. Para ello vamos a proponer una serie de plugins tanto dinámico como estáticos basados en el lenguaje de programación Java.

5.1.4 Análisis estático del código (SAST)

Es un análisis preliminar del código fuente, el cual permite a los desarrolladores identificar fallos técnicos o lógicos en el binario. Así como también a encontrar vulnerabilidades de seguridad sin necesidad de generar un deploy y ejecutar el código. A continuación mostraremos las herramientas y plugins que utilizaremos en este trabajo:

5.1.5 Análisis dinámico del código (DAST)

En un análisis de caja negra que intenta descubrir y analizar las aplicaciones en busca de vulnerabilidades. Generalmente lanzando peticiones y analizando la respuesta de la aplicación ya que el objetivo principal es comprobar los resultados para determinar si funciona correctamente a este tipo de análisis se le conoce como de caja negra.

Plugin	Lenguaje	Licencia
FindBug	Java	OpenSource
Sonarqube	Java,JS,C/C++...	OpenSource

TAULA 4: PLUGINS DE TIPO SAST

A continuación mostraremos las herramientas y plugins que utilizaremos en este trabajo:

Plugin	Lenguaje	Licencia
Grabber	Java	OpenSource
Sonarqube	Java,JS,C/C++...	OpenSource

TAULA 5: PLUGINS DE TIPO DAST

5.1.6 Contenedores

Como el mismo Título del proyecto lo indica la herramienta de contenedores que utilizaremos es **Docker** [9] es una plataforma que tiene todo lo necesario para ejecutar una aplicación o software en cualquier entorno. De esta manera permite entregar el software de una manera más rápida y compartir el desarrollo entre varios equipos asegurándote de que se ejecutara.

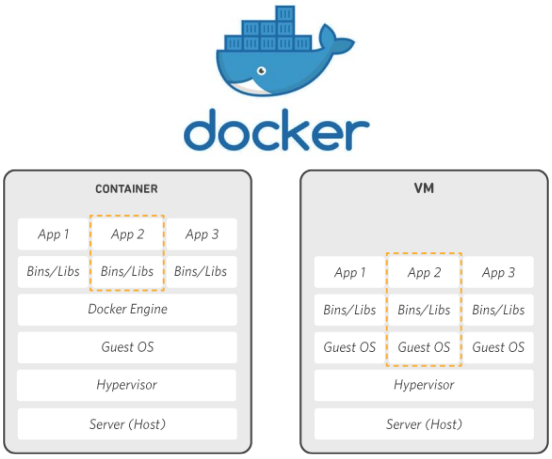

Fig. 7: Explicación de cómo funciona Docker[12].

Como podemos observar en la imagen, con Docker no necesitamos crear una máquina virtual por cada aplicación o software que ejecutamos simplemente le instalamos los binarios y librerías necesarias y se ejecuta sobre cualquier sistema operativo y equipo.

5.1.7 Repositorio

Sabemos que es esencial tener una herramienta de control de versiones o un lugar donde se guarde todo el código o trabajo que se realiza en el desarrollo y que esta pueda ser accesible a todo el equipo que trabaje en el proyecto, por

este motivo elegimos Git ya que es una de las herramientas más utilizadas por los desarrolladores.

Además se sincroniza muy bien tanto con Docker como Jenkins Pipeline y que puede coger el código directo del repositorio Git y generar un deploy.

Fig. 8: Logo de GitHub.

5.2 Ejecución de Resultados

5.2.1 Elección código de prueba

Para realizar las pruebas en el entorno práctico elegimos un código java basado en un modelo MVC, el modelo vista controlador consiste en separar el código en 3 modalidades diferentes por un lado los datos del programa en el package de model y el controlador que hace de enlace entre el modelo y la vista donde se encuentra la interfaz del usuario. Nuestro código crea una aplicación web utilizando java server faces. Mediante el portal web mostraremos ditas aleatorias cómicas de Dilbert [10]. Para acceder a este portal debemos acceder mediante login y password.

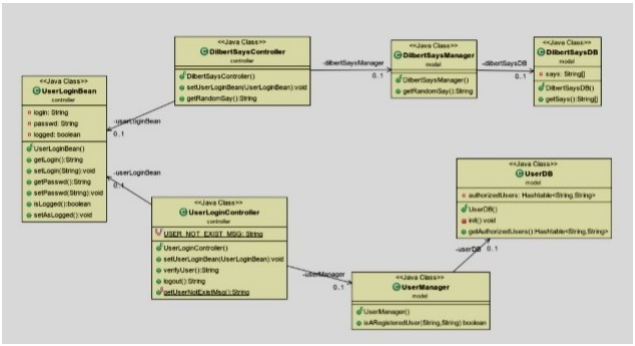

Fig. 9: Diagrama de clases del código Java.

Como podemos ver este código está compuesto por 7 clases java, 4 de ellas conforma el modelo y 3 forman parte del controlador.

5.2.2 Implementación del Entorno Práctico

En este apartado nos centraremos en el análisis del código, para ello vamos a instalar herramientas que nos permitan realizar esta acción.

Creación de un Pipeline en Jenkins

"Jenkins Pipeline es un conjunto de complementos que admite la implementación e integración de pipeline de entrega continua en Jenkins. [12]"

Para realizar la integración Continua es necesario crear una línea de instalación y análisis del código en **Jenkins**. Sincronizamos también nuestro pipeline al repositorio creado en GitHub donde tenemos el código.

5.2.3 Plugins

FindBug

"Un programa que utiliza análisis estático para buscar errores en el código Java. FindBugs [11]"

Hemos utilizado este plugin de manera local, para ello lo instalamos con el comando **apt-get install -y findbugs**. Para ejecutarlo solo debemos ejecuta el bash ./findbugs en el directorio de instalación.

SonarQube

"Es una plataforma de código abierto desarrollada por SonarSource para la inspección continua de la calidad del código para realizar revisiones automáticas con análisis estático del código para detectar errores, olores de código y vulnerabilidades de seguridad en más de 20 lenguajes de programación. [12]"

SonarQube ofrece varias formas de instalación para nosotros escogeremos las relacionadas con Docker.

Para ello necesitamos instalar Docker Compose [13], como su nombre mismo indica es un paquete de Docker que contiene más de una imagen. La cual te permite instalar y configurar un servicio en un solo paso a través del fichero **Docker-Compose.yml**. Con el comando **Docker-Compose up** instalamos estas imágenes.

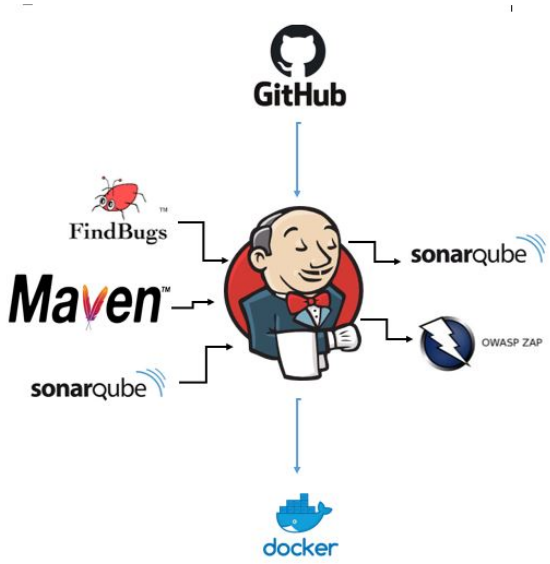

Fig. 10: Imagen gráfica de la integración en Jenkins.

Owasp Zap Scaning

"Es un escáner de seguridad de aplicaciones web de código abierto, está destinado a ser utilizado tanto por aquellos nuevos en seguridad de aplicaciones como por probadores profesionales de penetración.[14]"

Esta herramienta al igual que la de SonarQube, ofrece varias formas de instalación desde ejecutar un docker hasta la instalación local. En este caso decidimos por recursos realizar la instalación en local.

Descargamos el paquete de instalación desde la web oficial de Owasp, además también cuenta con una guía de inicio rápido y otras documentaciones[15].

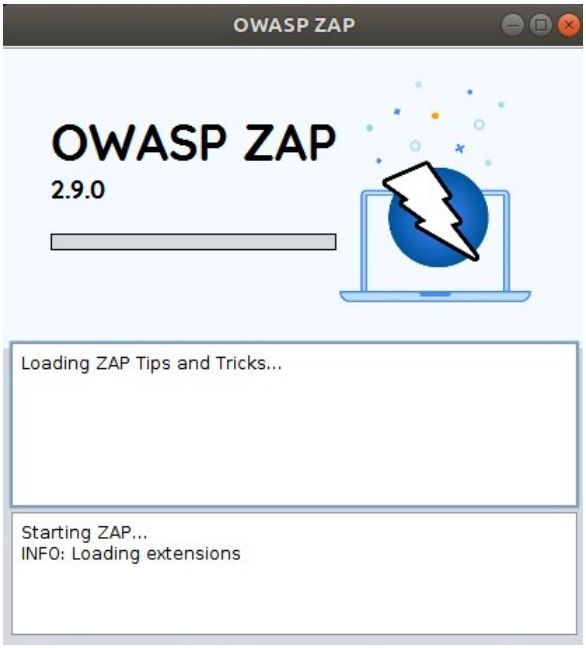

Fig. 11: Interfaz grafico de Owasp Zap.

5.2.4 Análisis del Código

Después de haber realizado la instalación de una parte de las herramientas a utilizar en el entorno práctico. Es necesario una evaluación de los resultados obtenidos por los diferentes plugins, así como también una comparativa entre ellos. En este apartado nos centraremos en los análisis de tipo SAST y DAST ejecutado sobre nuestro código. El resultado puede variar dependiendo del tipo de herramienta y plugin.

5.2.5 Análisis Estático de la Seguridad del Código (SAST)

FindBugs

En el apartado anterior, hemos explicado la instalación de esta herramienta tanto en Linux como en Eclipse. Para comenzar con el análisis primero creamos una archivo .war realizando un export de nuestro proyecto.
Creamos un nuevo proyecto, añadimos el archivo .war generado así como también las dependencias, ya que nuestro proyecto es un Dynamic Web Project que implementa java server faces. Configuramos las opciones para la realización del análisis.
Después de realizar las configuraciones y proceder al análisis podemos ver los siguientes resultados.

Plugin	Tipo	Error
Vulnerabilidad	Malicious Code	3
Errores de Codificación	JPerformance	1
Otros errores	Dodgy Code	4

TAULA 6: RESULTADOS DEL ANÁLISIS CON FINDBUG.

Como podemos visualizar en la tabla anterior tenemos 8 errores detectado por findbug.

SonarQube

Uno de los objetivos de este TFG, era realizar las pruebas con esta herramienta ya que se puede realizar tanto análisis de tipo SAST como DAST. En este caso hemos realizado 2 formas de test, la primera de ellas de manera local con un análisis de dependencias mediante **MAVEN** y la segunda desde la integración con jenkins mediante un plugin. Este último se explicara en el apartado 6.2.1

Para realizar el test de manera local a través del terminal de Linux. Debemos crear un proyecto en SonarQube para después generar un Token que nos permita ejecutar el análisis sin la necesidad de proporcionar nuestro usuario y contraseña[16].

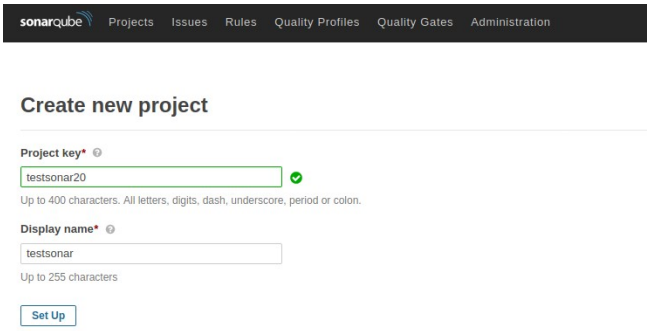

Fig. 12: Creando un Test para el Proyecto.

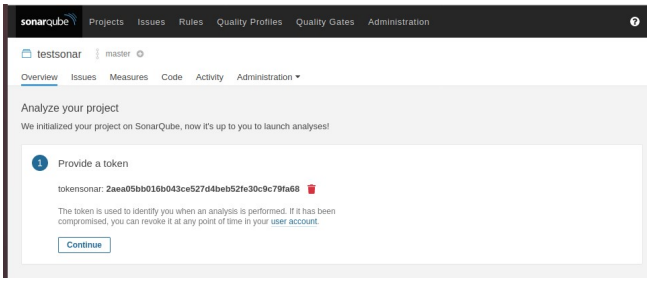

Fig. 13: Generando Token en SonarQube.

Elegimos el tipo de lenguaje de nuestro código que en nuestro caso es **Java** y para el análisis de dependencias escogemos **Maven** y nos generara los comandos a ejecutar en el terminal de Linux en el repositorio local de nuestro código. Procedemos a ejecutar el código anterior en

```
mvn sonar:sonar
-Dsonar.projectKey=testsonar20
-Dsonar.host.url=http://localhost:9000
-Dsonar.login=2aea05bb016b043ce527d4beb52fe30
```

TAULA 7: COMANDO LINUX PARA MAVEN

el repositorio local al que previamente hemos clonado mediante Git. Como podemos ver para SonarQube nuestro código no tiene ningún Bug ni tampoco vulnerabilidades de seguridad.

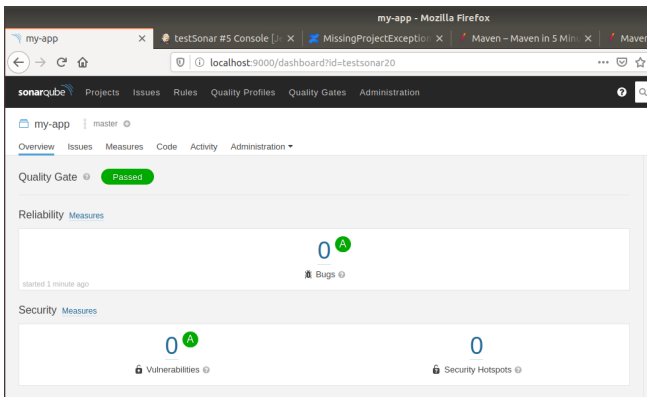

Fig. 14: Resultados en SonarQube.

5.2.6 Análisis Dinámico de la Seguridad del Código (DAST)

Owasp Zap Scaning Report

Esta herramienta nos permitirá realizar diferentes pruebas de seguridad de nuestro repositorio alojado en GitHub, además también permite realizar análisis a páginas web.
Owasp nos permite realizar las siguientes pruebas de seguridad[17]:

- Evaluación de vulnerabilidad.
- Pruebas de penetración.
- Pruebas de tiempo de ejecución.
- Revisión del código.

En este caso vamos a realizar los análisis de evalución de seguridad y revisión del código.

5.3 Integración Continua con Jenkins

Para realizar estas pruebas, tendremos que realizar una serie de configuraciones con el fin de utilizar algunas de las herramientas ya explicadas en este documento. Para poder obtener una integración siguiendo unas fases y definiendo pruebas antes y después de construir una release.

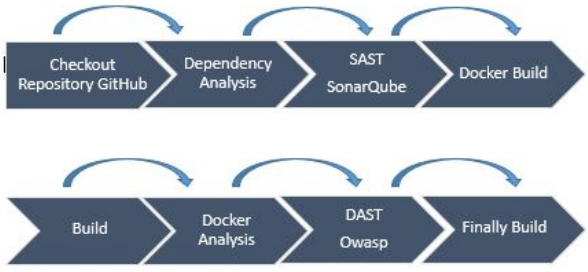

Fig. 15: Diagrama Lógico de las Fases de IC/DC.

5.3.1 Configuración de SonarQube en Jenkins

Para lograr el objetivo de obtener un pipeline de ejecución que garantice la integración continua, se han realizado muchas pruebas probando diferentes tipos de pipeline hasta encontrar uno que nos permita implementar todos los plugins necesarios.

Para ello creamos un item nuevo de tipo pipeline que nos permitirá mediante un fichero de configuración llamado JenkinsFile o Pipeline script estableciendo los parámetros necesarios para analizar el código y crear un ejecutable.

Como punto inicial, configuramos el entorno y la herramienta Maven.

Pipeline script

Script

```
pipeline {
    agent any
    enviroment {
        Workspace = pwd()
    }
    tools {
        maven 'Maven 3.6.3'
        jdk 'jdk8'
    }
    stages {
        stage ('Initialize') {
            steps {
                sh '''
                    echo "PATH = ${PATH}"
                    echo "M2_HOME = ${M2_HOME}"
                '''
            }
        }
    }
```

Fig. 16: Configuración de JenkinsFile.

Luego definimos el análisis de dependencias de nuestro repositorio en GitHub, podemos observar que se define la branca principal master para realizarlo.
Finalmente realizaremos el build del código en un contenedor Docker, para después realizar un análisis de seguridad del Docker generado. Por último y no menos importante sería el análisis dinámico una vez tengamos creado el build en el Docker. A continuación os mostramos el pipeline de ejecución que definimos a través del Jenkinsfile:

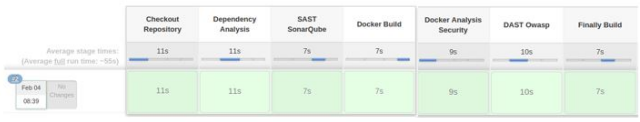

Fig. 17: Ejecución Pipeline de Jenkins.

6 Conclusiones

Durante el proceso de realización de este trabajo, hemos estudiado e investigado las metodologías ágiles y en concreto el modelo DevOps. ¿Por qué nace? ¿Como se utiliza y sus beneficios? son algunos de los conceptos reflejados en estas páginas. Así como también, establecer un ciclo de vida donde la seguridad adquiera un papel protagónico y se elimine la idea de que esta debe ser añadida al final y conseguir un desarrollo basado en la cultura SecDevOps.
Podríamos decir que hemos cumplido la mayoría de objetivos principales, que consistían en profundizar en el modelo DevOps, estudiar su situación actual y el impacto

en el ámbito de las TI. Cabe destacar que este proyecto esta más enfocado al estudio de este desarrollo y al uso de herramientas como Jenkins y Docker.

Con la ayuda de GitHub, algunos Plugins y las herramientas antes mencionadas hemos querido desarrollar un pipeline de ejecución e integración continua mediante Jenkins, incluyendo las fases por las cuales debe pasar un código o aplicación, de esta manera se consigue realizarlo de una forma automática para generar una nueva versión. Finalmente se ha necesitado mucho esfuerzo y horas de dedicación para conseguir los objetivos marcados, pero a la vez orgullosos por el trabajo realizado

TRABAJOS FUTUROS

Finalmente después de haber realizado todo el trabajo mencionado en este informe, solo nos queda plantear diferentes ramas de investigación que nos permita explorar otras técnicas, plugins y herramientas para el desarrollo de SecDevOps. Para poner en contexto los expresado con anterioridad mencionaremos algunas de las propuestas a realizar en el futuro:

- Utilizar plugins híbridos llamados IAST.
- Automatizar la ejecución de release en tiempo real.
- Realizar la integración continua a través de varios repositorios.
- Explorar otras herramientas utilizadas en DevOps.

AGRADECIMIENTOS

Primero agradecer a la gran comunidad de personas que mediante publicaciones en páginas web, videos de YouTube, foros y documentaciones aportadas en internet me han servido de gran ayuda para realización de este trabajo. También agradecer a Ángel Elbaz, al cual no he tenido el placer de tenerlo como profesor pero sí como Tutor y lo ha hecho extremadamente bien. Finalmente a mis grandes acompañantes de este viaje que han sido mis compañeros, no olvidaré esos fines de semana en la biblioteca de Sociales donde se mezclaba la impotencia, nervios y cansancio pero muchas otras de alegría y satisfacción cuando el trabajo daba sus frutos. Simplemente Gracias....

REFERÈNCIES

[1] ≪Agile Modeling (AM) Home Page: Effective Practices for Modeling and Documentation≫. [En línea]. Disponible en: http://agilemodeling.com/. [Accedido: 10-nov-2019].

[2] ≪Agile Alliance≫, Agile Alliance. [En línea]. Disponible en: https://www.agilealliance.org/. [Accedido: 10-nov-2019].

[3] ≪Continuous Integration≫, martinfowler.com. [En línea]. Disponible en: https://martinfowler.com/articles/ continuousIntegration.html. [Accedido: 10-nov-2019].

[4] ≪Actualizando DevOps a SecDevOps≫. [En línea]. Disponible en: http://www.decharlas.uji.es/es/actualizando-devops-a-secdevops. [Accedido: 10-nov-2019].

[5] ≪Broadcom Inc.Connecting Everything≫. [En línea]. Disponible en: https://docs.broadcom.com/docs/accelerating-velocity-and-customer-value-with-agile-anddevops-research-paper. [Accedido: 10-nov-2019].

[6] ≪Metodología Kanban Tool≫. [En línea]. Disponible en: https://kanbantool.com/es/metodologia-kanban. [Accedido: 29-sep-2019].

[7] ≪Trello. Qué es, Para Qué sirve y Cómo Funciona≫. [En línea]. Disponible en: https://www.expertosnegociosonline.com/que-es-trello-para-que-sirve/. [Accedido: 29-sep2019].

[8] ≪Pipeline≫, Pipeline. [En línea]. Disponible en: https://jenkins.io/doc/book/pipeline/index.html. [Accedido: 22-dic-2019].

[9] ≪Docker Hub≫. [En línea]. Disponible en: https://hub.docker.com/. [Accedido: 22-dic-2019]

[10] ≪Homepage — Dilbert by Scott Adams≫. [En línea]. Disponible en: https://dilbert.com/. [Accedido: 22-dic-2019].

[11] ≪FindBugsTM Find Bugs in Java Programs≫. [En línea]. Disponible en: http://findbugs.sourceforge.net/. [Accedido: 22-dic-2019].

[12] ≪SonarQube≫, Wikipedia. 03-oct-2019.

[13] ≪Get started with Docker Compose≫, Docker Documentation, 19-dic-2019. [En línea]. Disponible en: https://docs.docker.com/compose/gettingstarted/. [Accedido: 22-dic-2019].

[14] ≪OWASP ZAP≫, Wikipedia. 06-ago-2019.

[15] ≪Token de usuario SonarQube Docs≫. [En línea]. Disponible en: https://docs.sonarqube.org/latest/user-guide/user-token/. [Accedido: 22-dic-2019].

[16] ≪OWASP ZAP≫. [En línea]. Disponible en: https://www.zaproxy.org/. [Accedido: 05-feb-2020].

[17] ≪OWASP ZAP – Getting Started≫. [En línea]. Disponible en: https://www.zaproxy.org/getting-started/. [Accedido: 06-feb-2020].

APÉNDICE

A.1 Planificación

	FASE 1: Analisis de los Requerimientos y Estado del arte	12 días	lun 07/10/19	mar 22/10/19
	Recopilación y análisis de información	6 días	lun 07/10/19	lun 14/10/19
	Investigación del estado de arte	6 días	mar 15/10/19	mar 22/10/19
	FASE 2: Definición de la metodología y tecnologías	12 días	mié 06/11/19	jue 21/11/19
	Especificación de las fases de software seguro	5 días	mié 06/11/19	mar 12/11/19
	Aplicación y elección de técnicas en cada fase	7 días	mié 13/11/19	jue 21/11/19
	FASE 3: Creación del entorno de pruebas	7 días	jue 21/11/19	vie 29/11/19
	Descargar el software elegido	2 días	jue 21/11/19	sáb 23/11/19
	Instalación del Entorno Práctico	5 días	lun 25/11/19	vie 29/11/19
	FASE 4: Desarrollo e implementación del proyecto	31 días	lun 09/12/19	vie 17/01/20
	Escoger código para la demostración	7 días	lun 09/12/19	mar 17/12/19
	Implementación del código en el entorno práctico	15 días	mié 18/12/19	mar 07/01/20
	Testeo de integración correcta	9 días	mié 08/01/20	vie 17/01/20
	FASE 5: Fase de test, pruebas y corrección de errores	6 días	sáb 18/01/20	vie 24/01/20
	Comprobación de errores	3 días	sáb 18/01/20	mar 21/01/20
	Corrección de código	3 días	mié 22/01/20	vie 24/01/20
	FASE 6: Verificación de resultados finales	3 días	sáb 25/01/20	mar 28/01/20
	Validación de resultados	3 días	sáb 18/01/20	mar 21/01/20
	Creación de demo	2 días	lun 27/01/20	mar 28/01/20
	FASE 7: Documentación Informe Final	3 días	mié 29/01/20	vie 31/01/20
	FASE 8: Entrega memoria Final	7 días	vie 31/01/20	sáb 08/02/20
	Creación de Paper TFG	7 días	sáb 01/02/20	sáb 08/02/20

Fig. 18: Tareas con retraso en las fechas.

A.2 Elección Código de Prueba

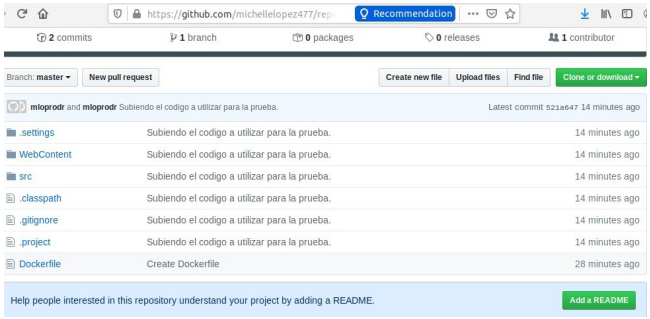

Fig. 19: Cdigo subido a GitHub.

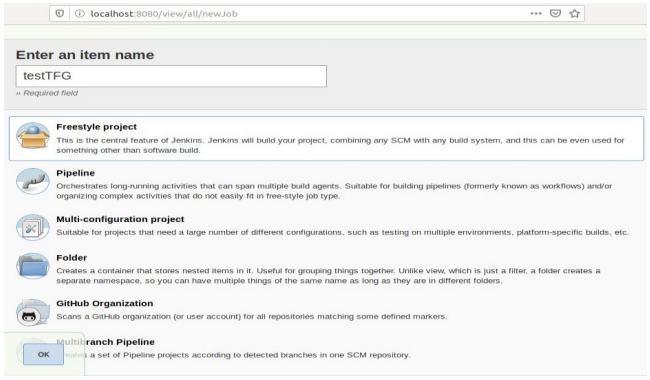

Fig. 20: Pipeline creado para el test.

A.3 Resultados

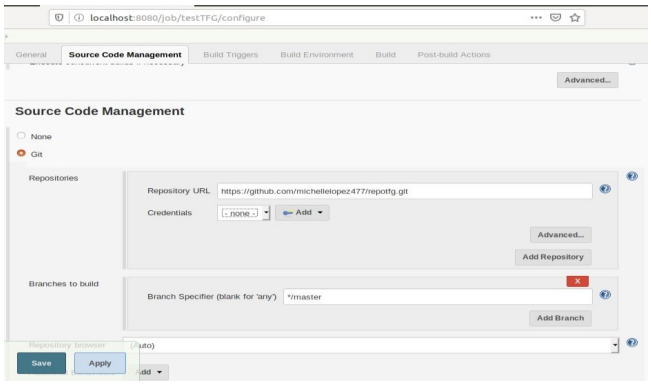

Fig. 21: Anadiendo Repositorio Git.

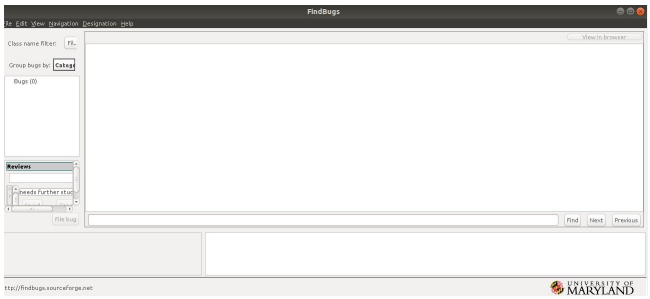

Fig. 22: Interfaz de FindBug.

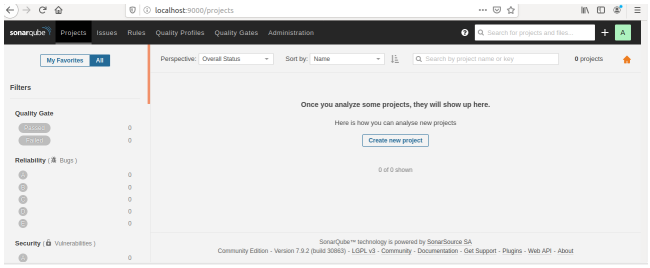

Fig. 23: Interfaz de SonarQube.

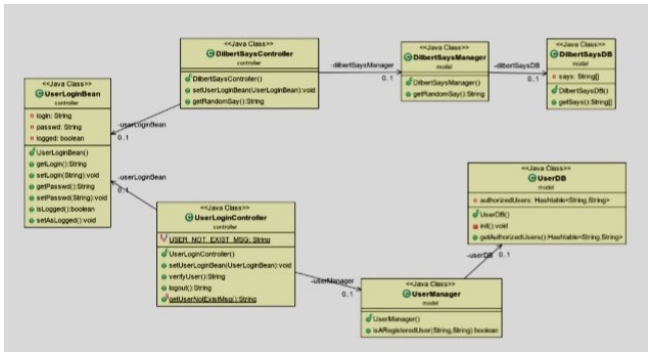

Fig. 24: Diagrama de clases del código Java.

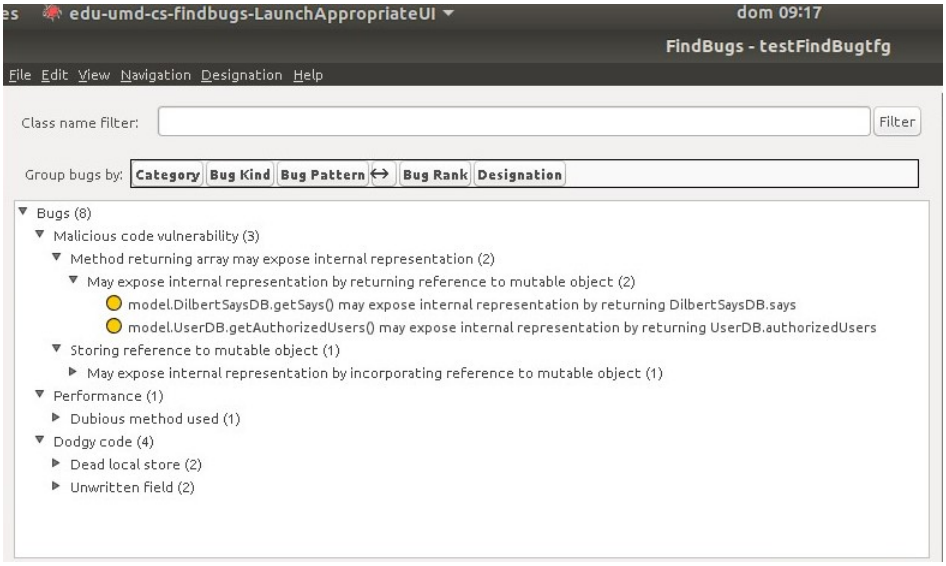

Fig. 25: Resultados del análisis con FindBug.

```
adminfg@ubuntu:~/repotfg$ mvn sonar:sonar \
>   -Dsonar.projectKey=testsonar20 \
>   -Dsonar.host.url=http://localhost:9000 \
>   -Dsonar.login=2aea05bb016b043ce527d4beb52fe30c9c79fa68
[INFO] Scanning for projects...
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-clean-plugin/2.5/maven-clean-plugin-2.5.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-clean-plugin/2.5/maven-clean-plugin-2.5.pom (3.9
kB at 2.5 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-plugins/22/maven-plugins-22.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-plugins/22/maven-plugins-22.pom (13 kB at 94 kB/
)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-parent/21/maven-parent-21.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-parent/21/maven-parent-21.pom (26 kB at 229 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/apache/10/apache-10.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/apache/10/apache-10.pom (15 kB at 141 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-clean-plugin/2.5/maven-clean-plugin-2.5.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-clean-plugin/2.5/maven-clean-plugin-2.5.jar (25
B at 224 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-install-plugin/2.4/maven-install-plugin-2.4.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-install-plugin/2.4/maven-install-plugin-2.4.pom
6.4 kB at 62 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-plugins/23/maven-plugins-23.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-plugins/23/maven-plugins-23.pom (9.2 kB at 107 k
/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-parent/22/maven-parent-22.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-parent/22/maven-parent-22.pom (30 kB at 275 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/apache/11/apache-11.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/apache/11/apache-11.pom (15 kB at 138 kB/s)
[INFO] Sensor HTML [web]
[INFO] Sensor HTML [web] (done) | time=21ms
[INFO] Sensor XML Sensor [xml]
[INFO] 1 source files to be analyzed
[INFO] 1/1 source files have been analyzed
[INFO] Sensor XML Sensor [xml] (done) | time=509ms
[INFO] ------------- Run sensors on project
[INFO] Sensor Zero Coverage Sensor
[INFO] Sensor Zero Coverage Sensor (done) | time=3ms
[INFO] SCM provider for this project is: git
[INFO] 1 files to be analyzed
[INFO] 0/1 files analyzed
[WARNING] Missing blame information for the following files:
[WARNING]   * pom.xml
[WARNING] This may lead to missing/broken features in SonarQube
[INFO] Calculating CPD for 0 files
[INFO] CPD calculation finished
[INFO] Analysis report generated in 199ms, dir size=74 KB
[INFO] Analysis report compressed in 7ms, zip size=10 KB
[INFO] Analysis report uploaded in 237ms
[INFO] ANALYSIS SUCCESSFUL, you can browse http://localhost:9000/dashboard?id=testsonar20
[INFO] Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
[INFO] More about the report processing at http://localhost:9000/api/ce/task?id=AW_DnCMXqF5whUHGZfTT
[INFO] Analysis total time: 10.031 s
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  24.643 s
[INFO] Finished at: 2020-01-20T07:39:47-08:00
[INFO] ------------------------------------------------------------------------
adminfg@ubuntu:~/repotfg$
```

Fig. 26: Ejecución testSonar en el terminal.

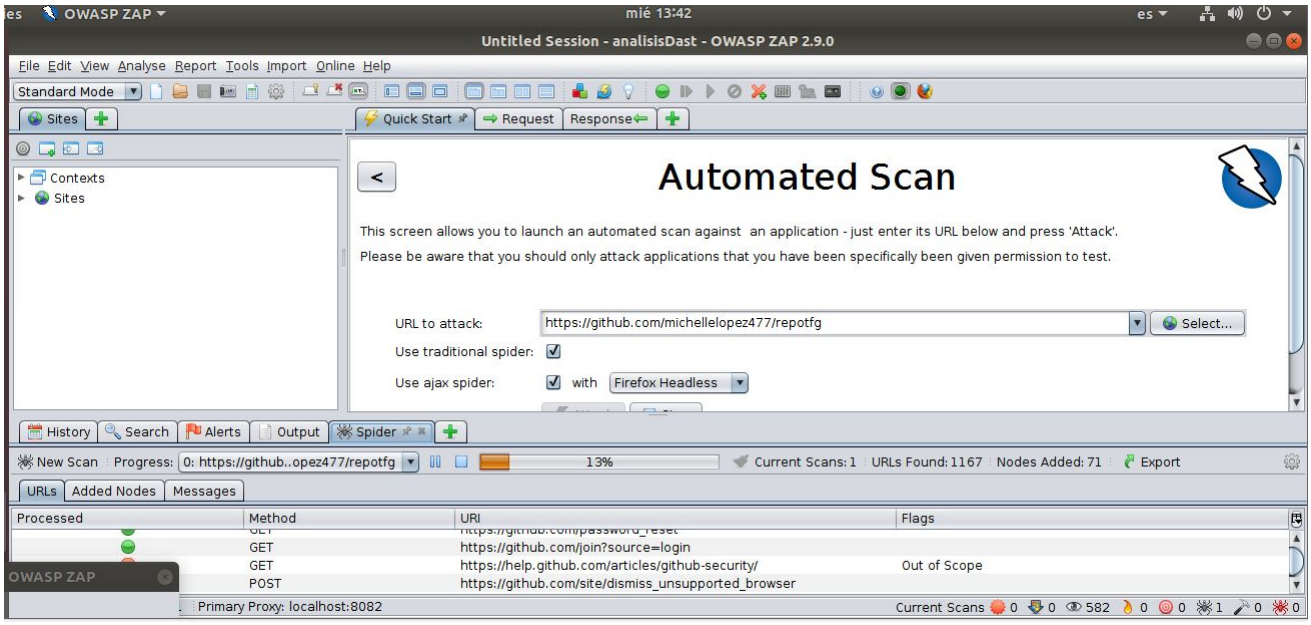

Fig. 27: Análisis con Owasp Zap.

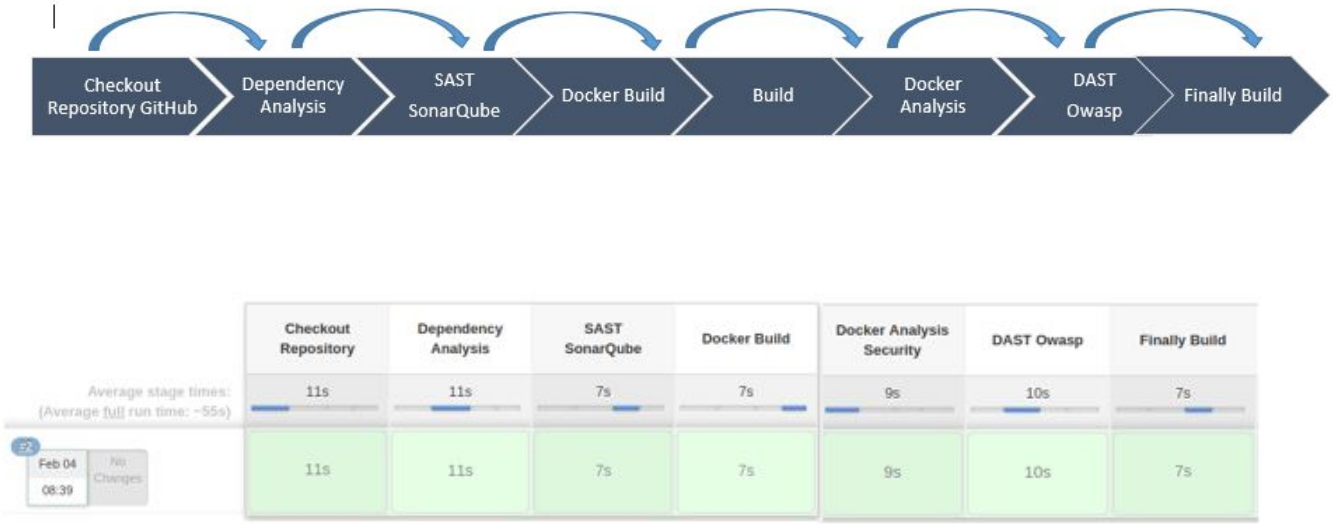

Fig. 28: Fases del pipeline de ejecución en Jenkins.

```
pipeline {
    agent any
    tools {
         maven "Maven 3.6.3"
         jdk "jdk8"
     }
     stages {
         stage ("Initialize") {
             steps {
                 sh 'echo "PATH = ${PATH}"'
                 sh 'echo "M2_HOME = ${M2_HOME}"'
                 sh 'mvn clean install -Dmaven.test.failure.ignore=true'
             }
         }
     }

    stages {
       stage("Check SCM") {
           steps{
             echo 'configuranso analisis'
             checkout([$class: 'GitSCM',
             branches: [[name: '*/master']],
             doGenerateSubmoduleConfigurations: false,
             extensions: [[$class: 'CleanCheckout']],
             submoduleCfg: [],
             userRemoteConfigs: [[credentialsId: 'passgit',
             url: 'https://github.com/user/repo.git']]])}
         }



       stage("SonarTest SAST Analysis") {
           steps{
             sh "/usr/local/bin/sonar-scanner" +
            ' -Dsonar.host.url=http://localhost:9000 ' +
            ' -Dsonar.sources=.'+
            ' -Dsonar.projectVersion=1.0'+
           ' -Dsonar.projectBaseDir=/var/lib/jenkins/workspace/projectTFG'+
           ' -Dsonar.projectKey=testsonar20'+
            ' -Dsonar.login=1e3757f08f63cd5a96c9f07f669258f5a3c26bee'+
            '-Dsonar.verbose=true'+
          echo 'AnĂ¡lisis Finalizado'
         }
       }
     }
     stages {
       stage("Docker Build") {
          steps {
            // Get some code from a GitHub repository
            echo 'Compiling...'
            sh "mvn install -f repotfg"
            echo 'Jar file generated'
            dir{'./repotfg'}{
                echo 'Building docker image'
                sh "docker build -t repotfg/repotfg-1.0"
                echo 'Docker image builded'}
            }

         }
         stage("Docker security Build") {
          steps {
```

Fig. 29: Código del fichero JenkinsFile.