

Trabajo Fin de Máster Ingeniería de Telecomunicación

Seguridad en la integración continua de la metodología ágil y la filosofía DevOps

Autor: Eleazar Rubio Sorrentino

Tutor: Juan Manuel Vozmediano Torres

**Dep. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla**

Sevilla, 2017



Trabajo Fin de Máster
Ingeniería de Telecomunicación

Seguridad en la integración continua de la metodología ágil y la filosofía DevOps

Autor:

Eleazar Rubio Sorrentino

Tutor:

Juan Manuel Vozmediano Torres

Profesor Titular

Dep. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Máster: Seguridad en la integración continua de la metodología ágil y la filosofía DevOps

Autor: Eleazar Rubio Sorrentino
Tutor: Juan Manuel Vozmediano Torres

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Apartado de agradecimientos.

Eleazar Rubio Sorrentino

Sevilla, 2017

Resumen

En los últimos tiempos de las empresas dedicadas al desarrollo de software como servicio (SaaS), los conceptos de metodología ágil y la filosofía Development and Operations (DevOps) están cobrando, cada vez más, un papel fundamental para el desarrollo de las mismas[12]. Según un estudio de alcance mundial realizado recientemente por CA Technologies, más del 75 por ciento de las organizaciones españolas coinciden en que las metodologías ágiles y DevOps son cruciales para el éxito de la transformación digital[20].

Este nuevo modo de entender el mundo del desarrollo de software (SW) posee una serie de elementos comunes, cada uno de ellos implementado con herramientas cada vez más conocidas y populares para las empresas que lo llevan a la práctica:

- Plataformas de desarrollo colaborativo y control de versiones de SW (por ejemplo GitHub), donde se almacena el código desarrollado por las mismas.
- Diferentes entornos o infraestructuras de trabajo para los desarrolladores, que van a permitir un desarrollo y despliegue continuo para las mejoras del producto: entornos de desarrollo, entornos de seguro de calidad o Quality Assurance (QA), preproducción, producción o entorno final, etc.
- Fundamentos de Integración continua (IC) y Despliegue Continuo (DC).
- Plataformas basadas en el despliegue de contenedores (generalmente Docker) que encapsulan los distintos elementos que componen el producto final y optimizan los recursos utilizados en las máquinas que los contienen, en su mayoría subcontratadas a terceras compañías (Amazon Web Services, Microsoft Azure, etc.).

El presente Trabajo Fin de Máster (TFM) pretende exponer un proceso automatizado de análisis estático de aplicaciones en varios niveles. El resultado de la ejecución periódica de estos procedimientos genera informes de seguridad que podrán ser utilizados para prevenir amenazas durante las fases más tempranas del Ciclo de Desarrollo Seguro (CDS) del software, advirtiendo de aspectos tales como:

1. Si la aplicación creada tiene Vulnerabilidades (Common Vulnerabilities and Exposures (CVE)) en las librerías de dependencias de código utilizadas.
2. Si la imagen que se va a emplear para desplegar el contenedor de dicho SW contiene vulnerabilidades conocidas al nivel de Sistema Operativo (SO).

TODO - Cambiar enlaces por referencias bibliográficas. Deployment and orchestration of immutable application images using containers (mostly Docker) that increases the portability, reusability and scalability

of the application. These applications use Cloud Providers as the underlying platform such as AWS, Azure or GCE among others. TRADUCIRLA EN EL PUNTO

Abstract

Lately inside Software as a Service (SaaS) companies, the agile methodology and DevOps philosophy concepts are taking a fundamental role for the development of them[12]. According to a recent global survey by CA Technologies, more than 75 percent of Spanish organizations agree that DevOps and agile methodologies are crucial to the success of digital transformation[20].

This new way of understanding the world of SW development has various common elements, each of them implemented with well known and popular tools for the companies that put it into practice:

- Collaborative development platforms and SW version control systems (for example GitHub), where the code developed is kept.
- Different environments and infrastructures for developers, which will allow to continuous development and deployment for product enhancements: development environments, Quality Assurance QA environments, pre-production, production or final environment, etc.
- Continuous Integration (CI) and Continuous Deployment (CD) principles.
- Deployment and orchestration of immutable application images using containers (mostly Docker) that increases the portability, reusability and scalability of the application. These applications use Cloud Providers as the underlying platform such as AWS, Azure or GCE among others.

This thesis tries to expose a procedure to automatically implement an application static analysis pipeline at several layers. The outcome of these periodically executed pipelines provide the company security reports that can be used to prevent threats during the earliest stages of the Secure Development Lifecycle (SDLC):

1. If the application created has Vulnerabilities (CVE) in the code dependencies used.
2. If the image used to deploy the container of the mentioned SW contains known vulnerabilities at the Operative System (OS) level.

Índice

<i>Resumen</i>	III
<i>Abstract</i>	V
1 Introducción	1
1.1 Contexto y motivación	1
1.2 Objetivo	2
1.3 Estructura de la memoria	3
2 Descripción de la Técnica	5
2.1 Metodología Ágil	5
2.2 Integración Continua IC y Despliegue Continuo DC	6
2.3 Ciclo de Desarrollo Seguro de software (SDLC)	6
2.4 Análisis estático	7
2.5 Trabajos relacionados	8
3 Entorno de trabajo	9
3.1 Git y GitHub	9
3.2 Bundler-audit	10
3.3 Nsp	12
3.4 Docker	13
3.5 Clair y Clairctl	14
3.6 Jenkins CI	15
3.7 Slack	16
4 Desarrollo de la solución	19
4.1 Preparación del entorno de trabajo	19
4.2 Trabajando con Jenkins	30
4.3 Evaluación de la solución	30
4.4 Resultados obtenidos	30
5 Conclusiones	31
Apéndice A Repositorio de GitHub	33
<i>Índice de Figuras</i>	35
<i>Índice de Códigos</i>	37
<i>Referencias</i>	39
Glosario	41

1 Introducción

You can ask 10 people for a definition of DevOps and likely get 10 different answers.

DUSTIN WHITTLE, 2014
DEVELOPER ADVOCATE AT UBER DEVELOPER PLATFORM

En este primer apartado de la memoria se pretende realizar con el lector un recorrido a través del contexto que ha motivado el desarrollo del presente TFM para el Máster en Seguridad de la Información y las Comunicaciones de la Universidad de Sevilla, con el fin de aclarar el objetivo perseguido en su realización. Como conclusión al mismo, se definirá la estructura seguida durante la redacción, introduciendo cada uno de los apartados que se encontrarán a continuación.

1.1 Contexto y motivación

El año 2017, para empresas englobadas en todo tipo de sectores, está siendo el año de la transformación digital. Estos procesos de transformación exigen distintas formas de trabajo, más ágiles y colaborativas, con las que poder aplicar nuevas tecnologías que permitan conseguir los objetivos del negocio, en entornos que afrontan grandes desafíos culturales, organizativos y operativos e incluso pueden llegar a tener que lidiar con sistemas tecnológicos antiguos y casi obsoletos[2].

Es en este contexto donde el concepto DevOps empieza a sonar con más fuerza: el contexto de las metodologías ágiles.

De esta forma, DevOps es un concepto de trabajo, basada en el desarrollo de código, que usa nuevas herramientas y prácticas para reducir la tradicional distancia entre técnicos de programación y de sistemas, respondiendo a la necesidad experimentada por el sector tecnológico de dar una respuesta más rápida a la implementación y operación de aplicaciones. Este nuevo enfoque de colaboración que es DevOps permite a los equipos trabajar de forma más cercana, aportando mayor agilidad al negocio y notables incrementos de productividad.

Desde las pruebas de concepto hasta el lanzamiento, pasando por el *testing* y los entornos de prueba, todos los pasos involucrados requieren de la máxima agilidad posible (Figura 1.1), y eso pasa por integrar los procesos y los equipos de programación con los de sistemas[3].

Por otro lado, el concepto de contenedor de aplicación (aislamiento de espacio de nombres y gobernanza de recursos) a pesar de no ser un concepto novedoso, está cobrando cada vez más y más relevancia en el

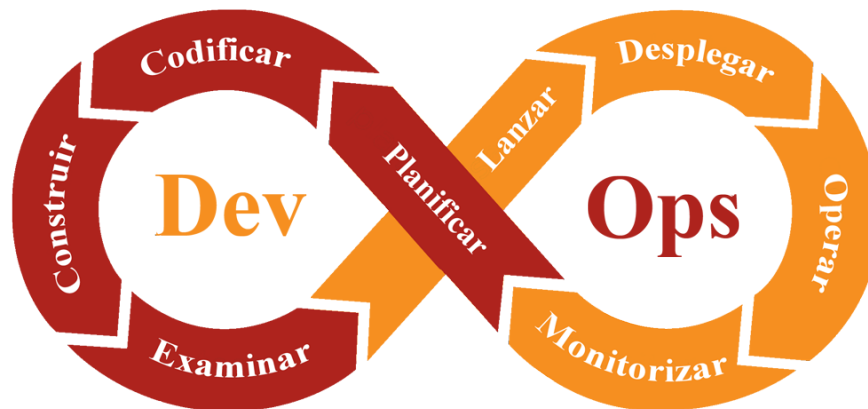


Figura 1.1 Introducción al proceso DevOps.

panorama de la empresa actual, de la mano de las continuas mejoras que experimentan las tecnologías que lo implementan, simplificando la administración y transformando la forma en que se desarrolla, distribuye y ejecuta el software, en forma de microservicio¹, además de proveer la habilidad de encapsular todo el entorno utilizado con el objetivo de ser desplegado en los sistemas de producción de la empresa, manteniendo las mismas características, aumentando la escalabilidad y disminuyendo notablemente los costes asociados a infraestructuras.

Los contenedores juegan un papel clave en un entorno DevOps porque soportan las implementaciones de la pila de desarrollo y operaciones completa y van en camino de formar parte de la definición básica de lo que se conocerá como DevOps en unos pocos años[6].

Además, la metodología DevOps representa una gran promesa a la hora de asegurar el desarrollo del software, ya que las organizaciones pueden potencialmente encontrar y remediar las vulnerabilidades con mayor frecuencia y al principio del ciclo de vida de la aplicación, ahorrando costes y tiempo. Conforme al informe de *"Seguridad de Aplicaciones y DevOps"* de octubre de 2016 promovido por Hewlett Packard Enterprise[7], que incluye tanto respuestas cualitativas como cuantitativas de profesionales de operaciones informáticas, líderes de seguridad y desarrolladores, se concluye que el 99% de los encuestados confirma que la adopción de la cultura DevOps aporta la oportunidad de mejorar la seguridad de las aplicaciones. Sin embargo, solo el 20% realizan análisis de seguridad de aplicaciones durante el desarrollo y el 17% no utilizan ninguna tecnología que proteja sus aplicaciones, destacando una desconexión significativa entre la percepción y la realidad de la seguridad DevOps.

Es en el contexto planteado donde surge la idea del presente TFM: aportar mecanismos a la metodología DevOps que permitan analizar la seguridad de las aplicaciones desarrolladas y el contenedor que las albergará dentro de la infraestructura de la empresa, sin interferir de manera destructiva con el propio proceso de desarrollo y despliegue de la aplicación.

1.2 Objetivo

El objetivo del presente Trabajo Fin de Máster (TFM) es desarrollar un entorno, basado en contenedores, que pueda ser incluido en el proceso de desarrollo e integración continua de la empresa y con el que poder realizar tareas periódicas programadas para analizar estáticamente las posibles vulnerabilidades (CVE entre otras) contenidas en las dependencias de aplicaciones desarrolladas mediante los lenguajes de programación

¹ Aproximación para el desarrollo software que consiste en construir una aplicación como un conjunto de pequeños servicios, los cuales se ejecutan en su propio proceso y se comunican con mecanismos ligeros (normalmente una API de recursos HTTP)

Ruby y NodeJS, además de analizar a nivel del SO vulnerabilidades presentes en las imágenes que van a constituir el contenedor que dará soporte a dichas aplicaciones.

Como medio para alcanzar el objetivo planteado se va a hacer uso de una serie de aplicaciones y herramientas, entre las que cabe destacar las siguientes, que serán desarrolladas en los próximos apartados:

- GitHub[11]: Plataforma de desarrollo colaborativo y control de versiones de SW donde almacenar, entre otras, el código desarrollado y que será analizadp.
- Jenkins[13]: Software de Integración Continua (IC) con el que automatizar los trabajos periódicos de análisis estático a realizar.
- Docker[8]: Proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, con el que desplegar los distintos elementos requeridos.

El trabajo aquí presentado no tiene como objetivo innovar en la tecnología existente, sino por contra valerse de esta para aportar al futuro usuario una herramienta intuitiva y de fácil aplicación, resultado de la agrupación de otras utilidades, con la que poder desplegar con el mínimo esfuerzo el entorno aquí recopilado.

1.3 Estructura de la memoria

Para facilitar la lectura de la memoria actual, se cree conveniente presentar un resumen de cómo se estructuran los diferentes apartados que contiene.

En el apartado actual, Introducción, se presenta el TFM que se va a realizar, aclarando el objetivo perseguido, el contexto en que surge y los aspectos que motivaron su realización.

El apartado 2, Descripción de la técnica, se pretende dar a conocer, de manera objetiva, las características de la realidad representada, con rasgos tales como elementos que la componen, utilidad, etc. En concreto, el apartado comienza describiendo repasando el panorama actual en las empresas de software. A continuación se ...

El apartado 3, Entorno de trabajo, está dedicado a conocer las herramientas utilizadas para poder llevar a cabo el desarrollo y la implementación de la parte técnica de este TFM.

El apartado 4, titulado Desarrollo de la aplicación, es el apartado principal de la memoria. En él, se detalla el proceso a seguir durante el desarrollo e implementación de los objetivos presentados en este proyecto, comenzando preparación del entorno que se va a utilizar en el desarrollo, hasta llegar a presentar el Resultado final obtenido.

Por último, el apartado 5, está dedicado a las Conclusiones y evaluaciones surgidas del proyecto, los objetivos alcanzados y las líneas futuras de trabajo surgidas durante la realización de éste.

2 Descripción de la Técnica

A good DevOps organization will free up developers to focus on doing what they do best: write software.

ROB STEWARD, 2015
GLOBAL VICEPRESIDENT AT VERINT-SYSTEMS

Para comprender el desarrollo del trabajo aquí presentado, tal y como se ha llevado a cabo, se debe conocer la situación en que éste se desarrolla, la tecnología de la que se dispone y los elementos existentes y necesarios, de una manera objetiva.

Es por esto, que el apartado actual está orientado a conocer las características de la realidad representada y a introducir las bases tecnológicas del presente TFM, resaltando los conceptos más importantes.

2.1 Metodología Ágil

La tecnología actual avanza a una velocidad considerable, provocando a su paso la renovación de la gestión de proyectos informáticos, debiendo esta alcanzar la velocidad de los cambios ocasionados por esta aceleración.

Así, la calidad, eficiencia, rapidez y flexibilidad en la entrega de un determinado producto se ha convertido en prioritaria, dando paso a la conocida como **Metodología Ágil**.

La Metodología Ágil envuelve un enfoque para la toma de decisiones en los proyectos software que plantean métodos de ingeniería del software basado en el desarrollo iterativo e incremental, donde los y soluciones evolucionan con el tiempo según la necesidad del proyecto. Así el trabajo es realizado mediante la colaboración de equipos auto-organizados y multidisciplinarios, inmersos en un proceso compartido de toma de decisiones a corto plazo[22].

La Figura 2.1 muestra el ciclo de vida en cada iteración de esta metodología.

El uso de procesos ágiles reporta los siguientes beneficios:

- Flexibilidad en el proceso y las definiciones de los productos.
- Realimentación continua con el cliente.



Figura 2.1 CAMBIAR LA IMAGEN.

- Iteración constante del producto, que se va analizando a medida avanza.
- Calidad mejorada.

2.2 Integración Continua IC y Despliegue Continuo DC

Integración continua IC es una práctica de desarrollo que requiere que los desarrolladores integren nuevos cambios en el código de la aplicación varias veces en un sólo día. Cada vez que esto ocurre, la inserción es verificada por una compilación automática, permitiendo a los equipos de trabajo implicados en el proceso detectar cualquier problema de forma temprana.

Integrando código regularmente los errores pueden ser detectados rápidamente y corregidos con más facilidad, debido a que al tratarse de un proceso frecuente la búsqueda del error va a quedar muy acotada.

El Despliegue Continuo DC está estrechamente relacionado a la IC y está referido a la liberación en los entornos de producción de la compañía de SW que está continuamente siendo probado de manera automática.

Adoptar ambos conceptos (IC y DC) no solo reduce los riesgos y permite una localización temprana de fallos de código, sino que también permite aumentar la velocidad de trabajo con el SW[23].

2.3 Ciclo de Desarrollo Seguro de software (SDLC)

Un ciclo de Desarrollo Seguro de software (en inglés, Software Development Life Cycle SDLC) es un marco de trabajo que define el proceso utilizado por las compañías a la hora de construir una aplicación desde sus inicios hasta el desmantelamiento de la misma. A lo largo de los últimos años, han surgido multitud

de modelos para SDLC, que han sido utilizados de diversas maneras acorde a las circunstancias de cada aplicación o empresa en general. Las fases comunes a todos estos modelos para el Ciclo de Desarrollo de software son las siguientes:

- Requisitos y planificación.
- Arquitectura y diseño.
- Planificación de las pruebas.
- Desarrollo del código.
- Pruebas y resultados.
- Lanzamiento y mantenimiento.

Con esto, un proceso de implementación de seguridad en el proceso SDLC garantizará que las actividades de seguridad, como las pruebas de penetración, la revisión del código y el análisis de la arquitectura, son parte integral del esfuerzo de desarrollo.

2.4 Análisis estático

static analysis (static code analysis)

Static analysis, also called static code analysis, is a method of computer program debugging that is done by examining the code without executing the program. The process provides an understanding of the code structure, and can help to ensure that the code adheres to industry standards. Automated tools can assist programmers and developers in carrying out static analysis. The process of scrutinizing code by visual inspection alone (by looking at a printout, for example), without the assistance of automated tools, is sometimes called program understanding or program comprehension.

The principal advantage of static analysis is the fact that it can reveal errors that do not manifest themselves until a disaster occurs weeks, months or years after release. Nevertheless, static analysis is only a first step in a comprehensive software quality-control regime. After static analysis has been done, dynamic analysis is often performed in an effort to uncover subtle defects or vulnerabilities. In computer terminology, static means fixed, while dynamic means capable of action and/or change. Dynamic analysis involves the testing and evaluation of a program based on execution. Static and dynamic analysis, considered together, are sometimes referred to as glass-box testing.

[16]

Análisis estático de software es un tipo de análisis de software que se realiza sin ejecutar el programa (el análisis realizado sobre los programas en ejecución se conoce como análisis dinámico de software).¹ En la mayoría de los casos, el análisis se realiza en alguna versión del código fuente y en otros casos se realiza en el código objeto. El término se aplica generalmente a los análisis realizados por una herramienta automática, el análisis realizado por un humano es llamado comprensión de programas (o entendimiento de programas) como también revisión de código.

2.4.1 Ruby y NodeJS

YAML no e stá por ningún lado.

Quizás a estas alturas no sea necesario hablar de estos lenguajes de programación... lo que si que daré será algunos datos que confirmen un poco por qué he empezado con estos...

Breve presentación a la importancia de estos lenguajes de programación, comentando que lo que se ha hecho aquí es extensible a otros lenguajes, con otras herramientas similares.
Concepto dependencias de código.

Bundler is the de facto way of managing dependencies. It provides, among other things, a clear way of specifying required libraries and their versions, by keeping track of everything for you through Gemfile and (for applications) Gemfile.lock. Exactly the sort of information you'd need when checking for security vulnerabilities.

2.4.2 De las dependencias del código

¿Qué es y cómo funciona?

2.4.3 De contenedores de imágenes

Quizás no son necesarios los subapartados... y simplemente baste con explicar el concepto de análisis estático de algo genérico.

<https://www.linuxadictos.com/docker-i-que-es-conociendo-la-ballena.html>

<http://www.javiergarzas.com/2015/07/que-es-docker-sencillo.html>

<https://www.redeszone.net/2016/02/24/docker-funciona-la-virtualizacion-contenedores/>

La nube es cada vez más grande, más potente, cuenta con más usuarios que hacen uso de ella al mismo tiempo y, además, permite la ejecución de aplicaciones cada vez más potentes, por lo que, para garantizar el correcto funcionamiento de esta, tanto en el presente como en el futuro, es necesario utilizar una plataforma que optimice los recursos lo mejor posible y, al mismo tiempo, sea lo más escalable posible con el fin de poder ampliar sus características de forma sencilla cuando sea necesario.

La nube es sinónimo de virtualización. Ejecutar un sistema operativo virtual por cada instancia de una aplicación es un proceso muy pesado y poco optimizado, a la vez que lento. Por ello, la comunidad Linux ha trabajado en el concepto de contenedores, una nueva forma de optimizar recursos creando pequeños espacios virtuales de las aplicaciones necesarias cargando solo el núcleo de la aplicación y las dependencias, pero funcionando siempre sobre un único kernel, o sistema operativo. (Esquema de esta WEB)

2.5 Trabajos relacionados

3 Entorno de trabajo

Technology is nothing. What's important is that you have a faith in people, that they're basically good and smart, and if you give them tools, they'll do wonderful things with them.

STEVE JOBS, 1994
BUSINESSMAN

Antes de comenzar el proceso de desarrollo de la aplicación es necesario preparar un entorno adecuado de trabajo, es decir, un conjunto de herramientas hardware y software que permitan llevar a cabo el proyecto con la mayor comodidad y precisión posible.

La correcta elección de un entorno de trabajo adecuado es fundamental a la hora de abordar cualquier tipo de proyecto, ya que el éxito o fracaso, o al menos la eficiencia del proceso de desarrollo del mismo, va a depender en gran medida de dicho entorno utilizado. El apartado actual presenta el entorno de trabajo utilizado para la realización de este TFM.

3.1 Git y GitHub

Los sistemas de control de versiones son programas cuyo principal objetivo es controlar los cambios producidos en el desarrollo de cualquier tipo de software, permitiendo conocer el estado actual de un proyecto, las personas que intervinieron en ellos, etc. Un buen control de versiones es tarea fundamental para la administración de un proyecto de desarrollo de software en general[1]. Git es uno de los sistemas de control de versiones más populares entre los desarrolladores, es gratuito, open source, rápido y eficiente, aunque gran parte su popularidad es debido a GitHub(Figura 3.7), un excelente servicio de alojamiento de repositorios de software que ofrece un amplio conjunto de características de gran utilidad para el trabajo en equipo.

A continuación se muestran algunas de las características que han llevado a GitHub a ser tan valorado entre los desarrolladores[15]:

- Permite versionar el código, es decir, guardar en determinado momento los cambios realizados sobre un archivo o conjunto de archivos con la oportunidad de tener acceso al historial de cambios al completo, bien para regresar a alguna de las versiones anteriores o bien para poder realizar comparaciones entre ellas.
- Gracias a la gran cantidad de repositorios de SW públicos que alberga, es posible leer, estudiar



Figura 3.1 Logotipo de GitHub.

y aprender de el código creado por miles de desarrolladores en el mundo, permitiendo incluso la oportunidad de adaptarlos a las necesidades propias de cada desarrollador, sin alterar el original y realizando una copia o fork¹ de este.

- Tras haber realizado un fork de un proyecto y haber realizado algunos ajuste, introducido alguna mejora o arreglado algún problema que este pudiera contener, es posible integrar los cambios realizados al proyecto original (previa supervisión de su propietario, administrador o alguno de sus colaboradores), por lo que un repositorio puede llegar a ser construido mediante la contribución una gran comunidad de desarrolladores.
- GitHub posee un sistema propio de notificaciones con el que poder estar informado de lo que ocurre en torno a un repositorio concreto, ya sea privado a la compañía o público a la comunidad.
- GitHub trae incorporado un visor de código, mediante el cual (y a través del navegador) es posible consultar el contenido de un archivo determinado, con la sintaxis correspondiente al lenguaje utilizado y sin necesidad de descargar una copia del mismo.
- Cada repositorio de SW albergado en GitHub cuenta con su propio seguimiento de incidencias, con un elaborado sistema de tickets, de manera tal que cualquier colaborador (o usuario en general) pueda reportar algún problema encontrado en la utilización el código o pueda simplemente sugerir nuevas características para que sean implementadas.
- Al ser una plataforma web es totalmente independiente al SO utilizado, siendo por otro lado Git compatible con los principales sistemas actuales: Linux, Windows, OSX.
- GitHub es gratuito e ilimitado para repositorios de proyectos públicos, sólo aquellos usuarios que deseen mantener proyectos en privado deberán pagar una cuota.

TODO - Una frase de cierre al apartado

3.2 Bundler-audit

Como ya fue comentado en el apartado 2 cada aplicación tiene sus dependencia, y estas a su vez pueden contener vulnerabilidades de seguridad. Encontrar las vulnerabilidades de seguridad que presenta una aplicación es una tarea necesaria y tediosa, que de ser obviada no impedirá que la aplicación generada siga siendo ejecutada como si todo estuviera funcionando en perfectas condiciones, pero que ocultará agujeros en la aplicación que podrán ser utilizados en diversa manera por algún usuario malintencionado.

Para cualquier aplicación escrita con Ruby y en ausencia de alguna herramienta automatizada de análisis de vulnerabilidades, el desarrollador del código deberá estar suscrito a cada lista de correo relacionada

¹ Copia exacta en crudo del repositorio original que podrá ser utilizada como un repositorio git cualquiera

con anuncios de seguridad de dependencias y realizar un seguimiento exclusivo de vulnerabilidades y actualizaciones de seguridad de cada una de las dependencias incluidas en la aplicación, para cada aplicación en la que participe[14].

Por el contrario, todo este proceso puede ser automatizado en Ruby gracias a `bundler-audit`[17] del grupo de colaboradores Rubysec, un verificador a nivel de parche para Bundler con las siguientes características:

- `bundler-audit` analiza las vulnerabilidades en las versiones de las gemas contenidas en el archivo de dependencias *Gemfile.lock* de la aplicación.
- Analiza las fuentes de dependencias que puedan ser inseguras (<http://>).
- Permite especificar avisos de seguridad que serán ignorados en el análisis, bien por ser un riesgo asumido por el desarrollador, una vulnerabilidad ya conocida y en la que se está actualmente trabajando o cualquier otro motivo.
- No requiere de conexión a internet para cada ejecución que se realice del análisis.
- Funcionando cruzando la información de dependencias recogidas del fichero *Gemfile.lock* con una lista de vulnerabilidades conocidas[18], basada en información pública existente en bases de datos como CVE.

El código 3.1 muestra un ejemplo del resultado obtenido a la salida del terminal de comandos al realizar un análisis estático de vulnerabilidades con `bundler-audit` al archivo *Gemfile.lock* de un proyecto:

Código 3.1 Ejemplo de uso de `bundler-audit`.

```
$ bundle audit
Name: actionpack
Version: 3.2.10
Advisory: OSVDB-91452
Criticality: Medium
URL: http://www.osvdb.org/show/osvdb/91452
Title: XSS vulnerability in sanitize_css in Action Pack
Solution: upgrade to ~> 2.3.18, ~> 3.1.12, >= 3.2.13

Name: actionpack
Version: 3.2.10
Advisory: OSVDB-89026
Criticality: High
URL: http://osvdb.org/show/osvdb/89026
Title: Ruby on Rails params_parser.rb Action Pack Type Casting Parameter
      Parsing Remote Code Execution
Solution: upgrade to ~> 2.3.15, ~> 3.0.19, ~> 3.1.10, >= 3.2.11

Name: activerecord
Version: 3.2.10
Advisory: OSVDB-90072
Criticality: Medium
URL: http://direct.osvdb.org/show/osvdb/90072
Title: Ruby on Rails Active Record attr_protected Method Bypass
Solution: upgrade to ~> 2.3.17, ~> 3.1.11, >= 3.2.12

Name: activerecord
Version: 3.2.10
Advisory: OSVDB-89025
```

```

Criticality: High
URL: http://osvdb.org/show/osvdb/89025
Title: Ruby on Rails Active Record JSON Parameter Parsing Query Bypass
Solution: upgrade to ~> 2.3.16, ~> 3.0.19, ~> 3.1.10, >= 3.2.11

Unpatched versions found!

```

Además, `bundler-audit` permite actualizar la base de datos `ruby-advisory-db` y analizar el fichero `Gemfile.lock` desde el mismo comando, habilidad que resulta de gran utilidad para su ejecución en sistemas de IC, como muestra el código 3.2:

Código 3.2 Ejecutando `bundler-audit` tras la actualización de las vulnerabilidades conocidas.

```
$ bundle audit check --update
```

Por último, y como ya se ha mencionado con anterioridad en este apartado, es posible ignorar advertencias especificadas (código 3.3) por el usuario:

Código 3.3 Ignorar vulnerabilidades con `bundler-audit`.

```
$ bundle audit check --ignore OSVDB-108664
```

TODO - Cierre del apartado. de Rubyssec para proyectos Ruby

3.3 Nsp

Nsp es la principal herramienta de interfaz de línea de comandos de Node Security Platform y es una de las herramientas más utilizadas para monitorizar la seguridad en aplicaciones node, permitiendo auditar el archivo `package.json` de dependencias de la misma.

Emplazamiento de Imagen

Figura 3.2 Escaneo de vulnerabilidades en las dependencias de Node JS.

Su utilización es simple, permitiendo incluso ignorar vulnerabilidades que puedan estar siendo tratadas en el momento del análisis y realizando consultas contra una base de datos de vulnerabilidades mantenida

por Node Security Platform, que contiene información recopilada de múltiples fuentes (CVE entre ellas). Basta con `"nsp check"` para analizar las vulnerabilidades de la aplicación, recibiendo un resultado como el mostrado en la Figura 3.2.

3.4 Docker



Figura 3.3 Logotipo de Docker.

Docker (Figura 3.3) es una herramienta de virtualización diseñada para aportar beneficios a desarrolladores, testers y administradores de sistemas, creando contenedores ligeros y portables para que las aplicaciones puedan ser ejecutadas en cualquier máquina, con sólo tener docker instalado e independientemente del SO de la propia máquina que lo contenga[5]. Esta plataforma de código abierto hace uso de las funciones de aislamiento de recursos que provee el kernel de Linux para dar lugar a contenedores independientes, dentro de los cuales se ejecutará una única aplicación con sus respectivas dependencias, funcionando siempre con este único kernel, en lugar de virtualizar uno por cada contenedor o máquina virtual, como muestra la Figura 3.4.

Emplazamiento de Imagen

Figura 3.4 Pila de funcionamiento de docker.

Gracias al uso de docker dos desarrolladores no tendrán que, por ejemplo, preocuparse de la versión Java que cada uno tenga instalado en su propia máquina durante el desarrollo de la aplicación, ya que está podrá ser enviada de uno a otro en el interior de un contenedor, que funcionará de la misma manera en cualquiera de los dos entornos.

Además, virtualizar con docker ofrece una serie de ventajas respecto a hacerlo con máquinas virtuales convencionales[21]:

- **Portabilidad.** Los contenedores son portables, por lo que pueden ser trasladados fácilmente a cualquier otro equipo con Docker sin tener que volver a configurar nada.
- **Ligereza.** Al no virtualizar un sistema completo, sino solo lo necesario, el consumo de recursos es muy inferior.
- **Autosuficiencia.** Docker se encarga de todo de organizarlo todo, por lo que los contenedores tan solo deben tener lo necesario para que la aplicación funcione.

Un sistema de contenedores Docker lo componen principalmente 5 elementos fundamentales:

- **Demonio:** proceso principal de la plataforma.
- **Cliente:** binario que constituye la interfaz al usuario y le permite interactuar con el Demonio.
- **Imagen:** Plantilla utilizada para crear el contenedor de la aplicación que se va a ejecutar en su interior. Dicha plantilla recibe el nombre por defecto de *Dockerfile*.
- **Registros:** Directorios donde se almacenan las imágenes, tanto de acceso público como privado y similares a los utilizados por Linux, donde los usuarios publican sus propios contenedores de manera que aquellos otros usuarios que los necesiten puedan descargarlos directamente desde allí.
- **Contenedores:** Donde se almacena todo lo necesario (librerías, dependencias, binarios de la aplicación, etc.) para que la aplicación pueda ejecutarse de forma aislada.

Actualmente existen multitud de empresas que utilizan a docker como sistema de contenedores en sus centros de datos (Spotify, eBay, PayPal, etc.) y cuenta con el apoyo de grandes compañías de internet como Amazon o Google, lo que permite que Docker se encuentre en un proceso continuo de crecimiento y mejora.

3.5 Clair y Clairctl

Clair (Figura 3.5) es un proyecto de código libre desarrollado por CoreOS para el análisis estático de vulnerabilidades en contenedores de aplicaciones, que funciona de la siguiente manera[4]:



Figura 3.5 Clair.

1. A intervalos regulares, Clair realiza una ingesta de metadatos de vulnerabilidad desde un conjunto configurado de fuentes (entre ellos CVE) y los almacena en una base de datos PostgreSQL.
2. Un cliente configurado para utilizar la Interfaz de Programación de Aplicaciones (API) de Clair indexa y escanea sus imágenes de contenedores por capas, creando una lista de características presentes en la

imagen y almacenándolas en la base de datos.

3. El cliente solicita a través de la API a la base de datos las vulnerabilidades conocidas para una imagen en particular, correlacionando las vulnerabilidades y características de la imagen, de manera que esta no requiere un escaneo completo para cada petición.
4. En caso de que se produzcan actualizaciones para los metadatos de vulnerabilidad, se enviará una notificación que alertará a los sistemas de que se ha producido un cambio, siendo escaneada de nuevo la imagen la próxima vez que sea solicitado un informe.

Por otro lado, Clairctl es uno de los clientes más utilizados para alcanzar la API de Clair y se caracteriza por ser una herramienta ligera de interfaz de comandos y escrita en el lenguaje de programación desarrollado por Google, Go. Clairctl es capaz de hacer de puente entre registros de imágenes tales como Docker Hub, Docker Registry o Quay.io (registro de imágenes de CoreOS). La Figura 3.6 muestra el esquema resultado de utilizar ambas herramientas para el análisis estático de contenedores de aplicaciones.

Emplazamiento de Imagen

Figura 3.6 Esquema de escaneo y detección de vulnerabilidades en imágenes Docker.

3.6 Jenkins CI

Jenkins CI (Figura 3.7) es una herramienta autónoma de código abierto que puede utilizarse para automatizar todo tipo de tareas, como la construcción, prueba y despliegue de software. Jenkins puede ser instalado a través de paquetes de sistemas nativos, Docker, o incluso ser ejecutado de manera independiente en cualquier máquina con el entorno de ejecución de Java instalado[13].



Jenkins

Figura 3.7 IC con Jenkins.

Jenkins posee, entre otras, las siguientes ventajas:

- **Continuous Integration and Continuous Delivery:** As an extensible automation server, Jenkins can be used as a simple CI server or turned into the continuous delivery hub for any project.
- **Easy installation:** Jenkins is a self-contained Java-based program, ready to run out-of-the-box, with packages for Windows, Mac OS X and other Unix-like operating systems.
- **Easy configuration:** Jenkins can be easily set up and configured via its web interface, which includes on-the-fly error checks and built-in help.
- **Plugins:** With hundreds of plugins in the Update Center, Jenkins integrates with practically every tool in the continuous integration and continuous delivery toolchain. If a plugin does not exist, you can code it and share with the community.
- **Extensible:** Jenkins can be extended via its plugin architecture, providing nearly infinite possibilities for what Jenkins can do.
- **Distributed:** Jenkins can easily distribute work across multiple machines, helping drive builds, tests and deployments across multiple platforms faster.
- It is an open source tool with great community support.
- It provides continuous integration pipeline support for establishing software development life cycle work flow for your application.
- It also provides support for scheduled builds & automation test execution.
- You can configure Jenkins to pull code from a version control server like GitHub, BitBucket etc. whenever a commit is made.
- It can execute bash scripts, shell scripts, ANT and Maven Targets.
- It can be used to Publish results and send email notifications.

Pipeline - A user-defined model of a continuous delivery pipeline, for more read the Pipeline chapter in this handbook.

Debo poner también la imagen de una pipeline de trabajo con verdes y rojos.

3.7 Slack

Slack (Figura 3.8) es una herramienta de mensajería en tiempo real diseñada para usarla en grupo, facilitando la gestión de los propios grupos de trabajo dentro del entorno de una empresa y centralizando la comunicación interna a la misma[19]. Además, permite la creación de grupos de trabajo con un número ilimitado de miembros, a la vez que es posible establecer canales de comunicación públicos, conversaciones privadas y compartir ficheros.

Por otro lado, Slack permite a sus usuarios trabajar con otros servicios basados en la web, amplificando aún más el trabajo colaborativo y proveyendo integración con múltiples herramientas de uso habitual: Dropbox, Trello, Google Drive, GitHub y Jenkins entre otras.

TODO - ¿Cierre al apartado?



Figura 3.8 Slack.

4 Desarrollo de la solución

*As an engineer, you learn there is a solution to every problem.
It may take you a while, but eventually you're going to find it.*

TONY CÁRDENAS
AMERICAN POLITICIAN

El apartado actual está dedicado a comprender el proceso seguido durante el desarrollo de la solución realizada. Comenzando con una introducción que sitúe al lector en condiciones que le permitan reproducir el proceso presentado y los primeros pasos a seguir en la preparación del entorno, desplegando los sistemas que componen dicha solución para así continuar automatizando el proceso en vistas a un uso prolongado en el tiempo. Por último, se evaluará la solución obtenida en un proceso de verificación de la solución mostrando el resultado final obtenido.

4.1 Preparación del entorno de trabajo

4.1.1 Docker

Como ya ha quedado reflejado en apartados anteriores, las nuevas habilidades proporcionadas por la virtualización mediante contenedores de aplicación permiten desplegar el sistema compuesto de múltiples contenedores con independencia al SO en el que estos se ejecuten, siendo el único requisito indispensable contar con el motor de virtualización de docker instalado y ejecutándose en la máquina. Por esto, toda solución desarrollada a partir de virtualización de contenedores comenzará con la instalación de docker en su sistema .

Por otro lado, se va a hacer uso de Docker compose, una herramienta de los creadores de Docker diseñada para permitir la definición, ejecución y seguimiento de aplicaciones Docker compuesta por múltiples contenedores. Será con la ayuda de Docker Compose como se despliegue el sistema utilizado.

No se cree necesario detallar en esta memoria el proceso seguido en la instalación de dichos elementos, ya que la propia documentación aportada para Docker[9] y Docker Compose[10] en los canales oficiales incluye una guía de como llevarla a cabo para cada SO, aunque es necesario comprender que cualquier reproducción del entorno desplegado comenzará siguiendo los pasos ofrecidos en dicha documentación.

4.1.2 Docker Compose

Esta sección tiene por objetivo especificar los elementos relevantes que conforman al archivo *docker-compose.yml*, necesario para poder automatizar la tarea de despliegue de todos los elementos. Para visualizar el contenido completo de este archivo, así como los archivos de configuración de los servicios que acompañan a este, se remite al lector al apéndice A (Repositorio de GitHub), donde encontrará el acceso a la ubicación de todos los ficheros utilizados para el desarrollo del presente TFM.

En primer lugar, el Código 4.1 muestra el fragmento de información que Docker Compose va a necesitar para ejecutar un contenedor (se va a usar Clair como ejemplo, por ser el más completo) que albergue a una aplicación. Cada elemento representa lo siguiente:

Código 4.1 Definiendo Clair en Docker Compose.

```
clair:
  container_name: clair_clair
  image: quay.io/coreos/clair:v2.0.0
  restart: unless-stopped
  depends_on:
    - postgres
  ports:
    - "6060-6061:6060-6061"
  links:
    - postgres
  volumes:
    - /tmp:/tmp
    - ./clair_config:/config
  command: [-config, /config/config.yaml]
```

- **container_name:** nombre que recibirá el contenedor una vez se encuentre en ejecución.
- **image:** ubicación de la imagen de Docker que va a ser desplegada. Este parámetro puede contener bien una referencia a un repositorio de imágenes Docker o bien a la ubicación del archivo *Dockerfile*, en la propia máquina, con las instrucciones para de creación de dicha imagen.
- **restart:** instrucciones de reinicio del contenedor ante imprevistos.
- **depends_on:** con la instrucción *depends_on* Docker Compose entenderá que el contenedor que se está desplegando depende de que exista otro (en el ejemplo actual postgres) en ejecución y no intentará ejecutar un contenedor sin la presencia del otro.
- **ports:** traducción de puertos "red_externa:red_docker_virtualizada" necesaria para poder acceder a dichos puertos desde el exterior al sistema Docker.
- **links:** permite conectar el contenedor con el servicio que se ejecuta en otro contenedor de forma que será accesible desde el hostname o el alias referenciado (https://postgres en el ejemplo actual).
- **volumes:** especifica la ruta o el volumen de Docker que será montado en el nuevo contenedor y la ruta destino a montar, en la forma "ruta_origen:ruta_destino".
- **command:** comando que ejecutará el contenedor una vez arrancado.

Visto lo anterior, el fragmento de Código 4.2 contiene las instrucciones necesarias para desplegar la base de datos PostgreSQL que necesitará Clair para su funcionamiento donde, por no ser alcance de este

proyecto, la contraseña que requerirá PostgreSQL para su funcionamiento está incluida como variable de entorno en el propio contenedor, suceso que bajo ningún concepto debe ocurrir en un sistema en producción. La documentación oficial de cada uno de los sistemas aquí tratados incluye información suficiente para reemplazar los métodos de conexión en claro por certificados SSL e intercambios de Tokens que permitan una conexión segura en un entorno distribuido.

Código 4.2 Definiendo PostgreSQL en Docker Compose.

```
postgres:
  container_name: clair_postgres
  image: postgres:9.6
  restart: unless-stopped
  volumes:
    - "data-postgres:/var/lib/postgresql/data"
  environment:
    POSTGRES_PASSWORD: password
```

Para el contenedor que contendrá Jenkins CI (Código 4.3) se necesitará además montar como volumen los binarios de Docker y el socket del demonio Docker, para proveer a Jenkins de la habilidad de ejecutar docker y ejecutar comandos del cliente Docker contra el resto de los contenedores desplegados:

Código 4.3 Definiendo Jenkins en Docker Compose.

```
jenkins:
  build: ./jenkins
  container_name: jenkins
  ports:
    - "80:8080"
  restart: "always"
  volumes:
    - "data-jenkins:/var/jenkins_home"
    - "reports:/reports"
    - "/usr/bin/docker:/usr/bin/docker"
    - "/var/run/docker.sock:/var/run/docker.sock:ro"
    - "/usr/lib/x86_64-linux-gnu/libltdl.so.7:/usr/lib/x86_64-linux-gnu/libltdl.so.7"
```

Por último, las herramientas de análisis estáticos podrían no ser definidas en el archivo *docker-compose.yml*, puesto que no son servicios de los que Docker Compose vaya a requerir mantener control constante, sino más bien aplicaciones clientes que serán ejecutadas en ciertos instantes de tiempo. Aún así, se ha decidido que incluir la definición de dichos contenedores entre la configuración del resto de los servicios facilita al usuario la tarea de despliegue inicial de las herramientas, debido a que será la propia herramienta Docker Compose la encargada de preparar en la máquina anfitriona las imágenes y volúmenes necesarios para la ejecución. El Código 4.4 muestra lo comentado.

Código 4.4 Definiendo Las herramientas de análisis estático en Docker Compose.

```
clairctl:
  build: ./clairctl
  container_name: clair_clairctl
  command: [echo, Hello from clairctl container]

ruby-tool:
  build: ./audit-tools/ruby
  container_name: ruby-tool
```

```

command: [echo, Hello from ruby-tool container]

nodejs-tool:
  build: ./audit-tools/nodejs
  container_name: nodejs-tool
  command: [echo, Hello from nodejs-tool container]

```

Una vez definidos todos los elementos, la ejecución del sistema se resume en la instrucción "docker-compose up -d" desde la carpeta contenedora del fichero *docker-compose.yml* en la máquina. Este comando mantendrá en ejecución y en segundo plano (-d) los servicios mencionados, creando las imágenes de Docker que pudiera necesitar para ello. La Figura 4.1 muestra la información aportada por el sistema en funcionamiento.

```

e1eazarr [~/Master_Ciberseguridad_US/Entorno] (master)$ docker-compose up -d
Creating network "entorno.default" with the default driver
Creating jenkins
Creating nodejs-tool
Creating clairctl
Creating ruby-tool
Creating c_postgres
Creating clair_clair
e1eazarr [~/Master_Ciberseguridad_US/Entorno] (master)$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED            STATUS              PORTS              NAMES
6d52c1f965cb       quay.io/coreos/clair:v2.0.0   "/clair -config /c..." 6 seconds ago      Up 5 seconds              0.0.0.0:6060-6061->6060-6061/tcp   clair_clair
872f9aa9619c       entorno_nodejs-tool           "echo 'Hello from ...'" 9 seconds ago      Exited (0) 7 seconds ago                               nodejs-tool
32e12221bb97       postgres:9.6                "docker-entrypoint..." 9 seconds ago      Up 7 seconds              5432/tcp              c_postgres
7aed6281f4d       entorno_clairctl             "echo 'Hello from ...'" 9 seconds ago      Exited (0) 7 seconds ago               clairctl
6dbe664d7699       entorno_ruby-tool            "echo 'Hello from ...'" 9 seconds ago      Exited (0) 6 seconds ago               ruby-tool
99e047e0cf6b       entorno_jenkins              "/bin/tini -- /usr..." 9 seconds ago      Up 8 seconds              50000/tcp, 0.0.0.0:80->8080/tcp   jenkins
e1eazarr [~/Master_Ciberseguridad_US/Entorno] (master)$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED            SIZE
entorno_ruby-tool    latest             5b70011c1e0        41 hours ago      118MB
entorno_jenkins      latest             5b70011c1e0        41 hours ago      844MB
entorno_nodejs-tool  latest             b830d5f9628e       41 hours ago      60.7MB
entorno_clairctl     latest             8dfd7cb15f62       41 hours ago      1.34GB
postgres             9.6                eb0f0e9af6d2       2 days ago        268MB
jenkins/jenkins      1fs                aca6348878dd       3 days ago        840MB
golang               1.8.3              ec1b36e59395       4 days ago        699MB
ruby                 2.1.10-alpine     614872f94ade       2 months ago      118MB
alpine               latest             7328f6f8b418       2 months ago      3.97MB
quay.io/coreos/clair v2.0.0            c5ec08ee85d5       3 months ago      387MB
node                 6.10.2-alpine     24d2680547f5       4 months ago      54.1MB
e1eazarr [~/Master_Ciberseguridad_US/Entorno] (master)$

```

Figura 4.1 Despliegue del entorno con Docker Compose.

4.1.3 Probando los componentes desplegados

Ahora que todos los sistemas están funcionando en el sistema de contenedores es momento de comprobar que la configuración proporcionada es la correcta y que el comportamiento es el esperado. Para ello, y antes de automatizar las tareas con la ayuda de Jenkins, se van a realizar desde el interfaz de comandos las instrucciones necesarias que confirmen que los análisis se están llevando a cabo sin problema por parte de las herramientas de análisis.

El Código 4.5 muestra la ayuda aportada por el comando "docker exec", que será la utilidad proporcionada por Docker para la ejecución de instrucciones en el interior de los contenedores que se encuentran en ejecución en el entorno (Jenkins, clair_clair y c_postgres).

Código 4.5 Comando de ayuda de docker exec.

```

$ docker exec --help

Usage: docker exec [OPTIONS] CONTAINER COMMAND [ARG...]

Run a command in a running container

Options:
  -d, --detach           Detached mode: run command in the background
  --detach-keys string   Override the key sequence for detaching a container
  -e, --env list         Set environment variables
  --help                 Print usage

```

<code>-i, --interactive</code>	Keep STDIN open even if not attached
<code>--privileged</code>	Give extended privileges to the command
<code>-t, --tty</code>	Allocate a pseudo-TTY
<code>-u, --user string</code>	Username or UID (format: <name uid>[:<group gid>])

Para las imágenes de contenedores que no estén en ejecución Docker incluye la utilidad "docker run", con multitud de opciones y argumentos que se adaptan a las ejecuciones más variadas que puedan requerir las imágenes del sistema. El Código 4.5 ejemplifica el uso de la instrucción que será necesario llevar a cabo en el entorno actual presentado.

Código 4.6 Comando de ayuda de docker exec.

```
$ docker run --help

Usage: docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

...

$ sudo docker run --rm -t -i -v ruta_local:ruta_contenedor imagen:tag /bin/bash
root@394bb66c4df0:/home/contenedor# pwd
/home/contenedor
```

Donde:

- `--rm` indica al demonio de Docker que una vez se termine de trabajar en el contenedor creado, este debe ser eliminado del sistema.
- `-t -i` sirve para especificar que se va a abrir un terminal (-t) interactivo (-i) con el contenedor.
- `-v` irá continuado por el volumen que se va a montar en el interior del contenedor de Docker.
- `imagen:tag` será la imagen utilizada para crear el contenedor.
- `/bin/bash` va a ser la instrucción ejecutada una vez creado el contenedor. Este comando va a devolver al usuario un terminal bash interactivo desde el interior del contenedor, donde poder ejecutar las instrucciones necesarias.

Conocidos estos comando y utilidades será posible comprobar la funcionalidad del sistema montado.

Bundler Audit

El Código 4.7 muestra los pasos a seguir para comprobar que Bundler Audit funciona correctamente desde el contenedor de Docker.

Código 4.7 Probando bundle-audit desde el contenedor.

```
eleazarr [~/Master_Ciberseguridad_US/Entorno] (master)$ docker run --rm -ti
entorno_ruby-tool:latest sh

/ # curl https://raw.githubusercontent.com/EleazarWorkshare/gemfile_vulnerable/
master/Gemfile.lock -o ./Gemfile.lock -s
/ # curl https://raw.githubusercontent.com/EleazarWorkshare/gemfile_vulnerable/
master/Gemfile -o ./Gemfile -s
```

```

/ # bundle-audit check
Name: activesupport
Version: 4.1.1
Advisory: CVE-2015-3226
Criticality: Unknown
URL: https://groups.google.com/forum/#!topic/ruby-security-ann/7VlB_pck3hU
Title: XSS Vulnerability in ActiveSupport::JSON.encode
Solution: upgrade to >= 4.2.2, ~> 4.1.11

Name: activesupport
Version: 4.1.1
Advisory: CVE-2015-3227
Criticality: Unknown
URL: https://groups.google.com/forum/#!topic/rubyonrails-security/bahr2JLnxvk
Title: Possible Denial of Service attack in Active Support
Solution: upgrade to >= 4.2.2, ~> 4.1.11, ~> 3.2.22

Vulnerabilities found!

```

Analizando la ejecución en detalle:

1. Se accede al contenedor que incluye la herramienta Bundle Audit.
2. Mediante la API de GitHub se descarga desde el repositorio que se quiere analizar los ficheros *Gemfile* y *Gemfile.lock* que incluyen las dependencias de la aplicación examinada.
3. "bundle-audit check" examina los ficheros, descubriendo que existen vulnerabilidades en la versión 4.1.1 de la dependencia *activesupport* contenida en el repositorio.

NSP

El funcionamiento de nsp en el terminal de comando es muy similar al de bundler-audit, como demuestra la Figura 4.2.

```

+ x Terminal
eleazarr [~/Master_Ciberseguridad_US/Entorno] (master)$ docker run --rm -ti entorno_nodejs-tool:latest sh
/ # apk update
fetch http://dl-cdn.alpinelinux.org/alpine/v3.4/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.4/community/x86_64/APKINDEX.tar.gz
v3.4.6-202-g827fd04 [http://dl-cdn.alpinelinux.org/alpine/v3.4/main]
v3.4.6-160-g14ad2a3 [http://dl-cdn.alpinelinux.org/alpine/v3.4/community]
OK: 5977 distinct packages available
/ # apk add curl
(1/4) Installing ca-certificates (20161130-r0)
(2/4) Installing libssh2 (1.7.0-r0)
(3/4) Installing libcurl (7.55.0-r0)
(4/4) Installing curl (7.55.0-r0)
Executing busybox-1.24.2-r13.trigger
Executing ca-certificates-20161130-r0.trigger
OK: 8 MiB in 17 packages
/ #
/ # curl https://raw.githubusercontent.com/EleazarWorkshare/package_json_vulnerable/master/package.json -o ./package.json -s
/ #
/ # nsp check
(*) 1 vulnerabilities found

```

	Tmp files readable by other users
Name	sync-exec
CVSS	4 (Medium)
Installed	0.6.2
Vulnerable	All
Patched	None
Path	package_vulnerable@0.0.0 > airbrake@1.3.0 > sync-exec@0.6.2
More Info	https://nodesecurity.io/advisories/310

Figura 4.2 Comprobando la utilidad nsp.

Con lo que queda demostrado que es posible desde el contenedor que ejecuta nsp acceder a la API de GitHub, descargar un archivo *package.json* con las dependencias de la aplicación node y realizar un análisis sobre este.

Clair y Clairctl

La forma de funcionamiento de Clairctl con Clair es diferente a las vistas para Bundler Audit y NSP, ya que Clairctl requiere de una configuración apropiada que le permita comunicarse con Clair y deberá realizar una serie de pasos para ello:

1. Enviará la imagen que se quiere analizar a Clair para que la almacene (push). Dicha imagen podrá encontrarse en la máquina local que contiene a Clairctl o en un repositorio de imágenes de internet (pull).
2. Clairctl pedirá a Clair que analiza la imagen enviada y le devuelva el resultado de dicho análisis (analyze).
3. Clairctl generará un informe en formato HTML que podrá ser visualizado en el navegador del usuario.

El proceso comentado se traduce en un mínimo de dos instrucciones de terminal para Clairctl: una para enviar y analizar la imagen en Clair y otra para realizar los informes. El Código 4.8 muestra el resultado de estas ejecuciones, tomando como ejemplo la propia imagen que contiene a Clairctl.

Código 4.8 Generando un informe HTML con Clairctl.

```
eleazarr [~/Master_Ciberseguridad_US/Entorno] (master)$ docker run --rm -ti -v
/home/eleazarr/informe_clairctl:/go/src/github.com/jgsquare/clairctl/
reports -v /var/run/docker.sock:/var/run/docker.sock:ro --net
entorno_default --link clair_clair:clair -p 57330:57330 entorno_clairctl
bash
root@59d0b5391fc6:/go# cd src/github.com/jgsquare/clairctl/
root@59d0b5391fc6:/go/src/github.com/jgsquare/clairctl# clairctl health

Clair:

root@59d0b5391fc6:/go/src/github.com/jgsquare/clairctl# ./clairctl analyze --
local entorno_clairctl:latest
2017-09-03 18:03:45.717118 I | config: retrieving interface for local IP
2017-09-03 18:03:45.717535 I | server: Starting Server on 172.20.0.4:57330
2017-09-03 18:03:45.722495 I | config: retrieving interface for local IP
2017-09-03 18:03:45.722730 I | clair: using http://172.20.0.4:57330/local as
local url
2017-09-03 18:03:45.722748 I | clair: Pushing Layer 1/9 [b3ddc1248fc3]
2017-09-03 18:03:46.651866 I | clair: Pushing Layer 2/9 [b7f7dd16a445]
2017-09-03 18:03:46.654461 I | clair: Pushing Layer 3/9 [85ac2ce41a64]
2017-09-03 18:03:46.656823 I | clair: Pushing Layer 4/9 [089137f68fc4]
2017-09-03 18:03:46.658893 I | clair: Pushing Layer 5/9 [382f1b17bd70]
2017-09-03 18:03:46.660915 I | clair: Pushing Layer 6/9 [fef71688ca46]
2017-09-03 18:03:46.662980 I | clair: Pushing Layer 7/9 [71e3be3b4746]
2017-09-03 18:03:46.664839 I | clair: Pushing Layer 8/9 [76ceb8887997]
2017-09-03 18:03:46.666559 I | clair: Pushing Layer 9/9 [5ee4b01f03bd]
2017-09-03 18:03:46.668400 I | config: retrieving interface for local IP
2017-09-03 18:03:46.668611 I | clair: using http://172.20.0.4:57330/local as
local url
2017-09-03 18:04:03.116083 I | clair: analysing layer [5ee4b01f03bd] 1/9
```

```

2017-09-03 18:04:03.143454 I | clair: analysing layer [76ceb8887997] 2/9
2017-09-03 18:04:03.190460 I | clair: analysing layer [71e3be3b4746] 3/9
2017-09-03 18:04:03.212756 I | clair: analysing layer [fef71688ca46] 4/9
2017-09-03 18:04:03.238765 I | clair: analysing layer [382f1b17bd70] 5/9
2017-09-03 18:04:03.285257 I | clair: analysing layer [089137f68fc4] 6/9
2017-09-03 18:04:03.302773 I | clair: analysing layer [85ac2ce41a64] 7/9
2017-09-03 18:04:03.316155 I | clair: analysing layer [b7f7dd16a445] 8/9
2017-09-03 18:04:03.325171 I | clair: analysing layer [b3ddc1248fc3] 9/9

Image: docker.io/entorno_clairctl:latest

Unknown: 15
Negligible: 65
Low: 49
Medium: 83
High: 37
Critical: 0
Defcon1: 0
root@59d0b5391fc6:/go/src/github.com/jgsquare/clairctl# ./clairctl report --
    local entorno_clairctl:latest

HTML report at ./reports/html/analysis-entorno_clairctl-latest.html

```

La Figura 4.3 muestra el resultado generado por Clairctl en el análisis de su propia imagen de Docker, confirmando que el sistema al completo se encuentra en perfecto estado de funcionamiento.

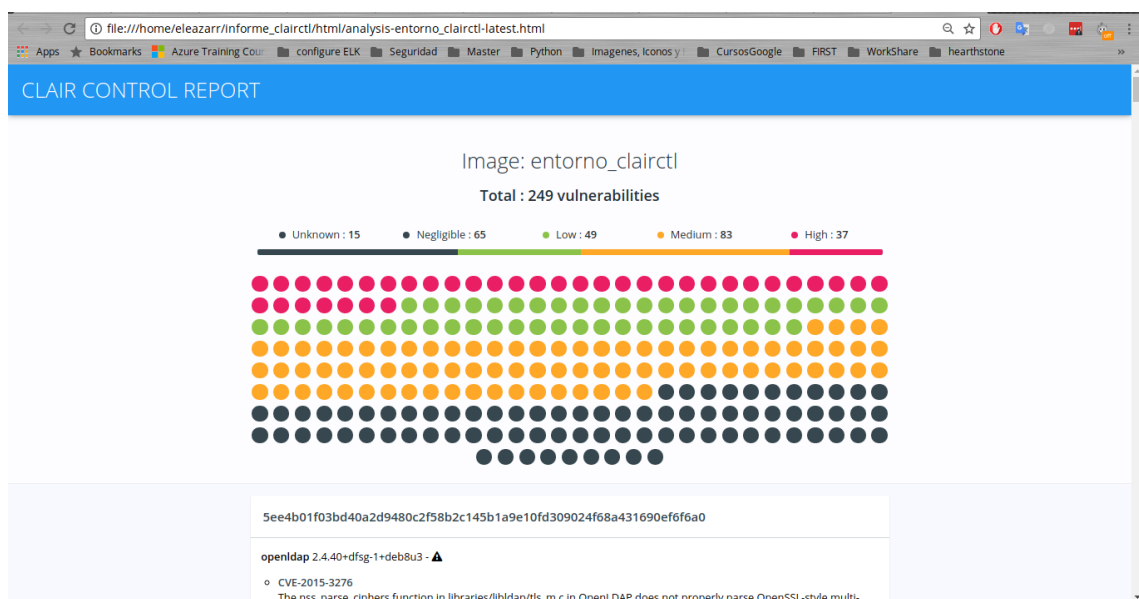


Figura 4.3 Informe generado por Clairctl.

Jenkins CI

Para concluir el apartado se va a confirmar que existe el acceso apropiado al panel de trabajo de Jenkins, así como que se dispone de los plugins necesarios instalados para poder automatizar los trabajos de Jenkins, más conocidos por su traducción al inglés Jenkins Jobs.

En primer lugar, la Figura 4.4 muestra el mensaje de bienvenida que el usuario encontrará la primera vez que accede a la URL de la aplicación, solicitando un código oculto en el sistema de archivos de la misma. La Figura 4.5 muestra el acceso al código necesario para comenzar a crear el primer usuario de Jenkins.

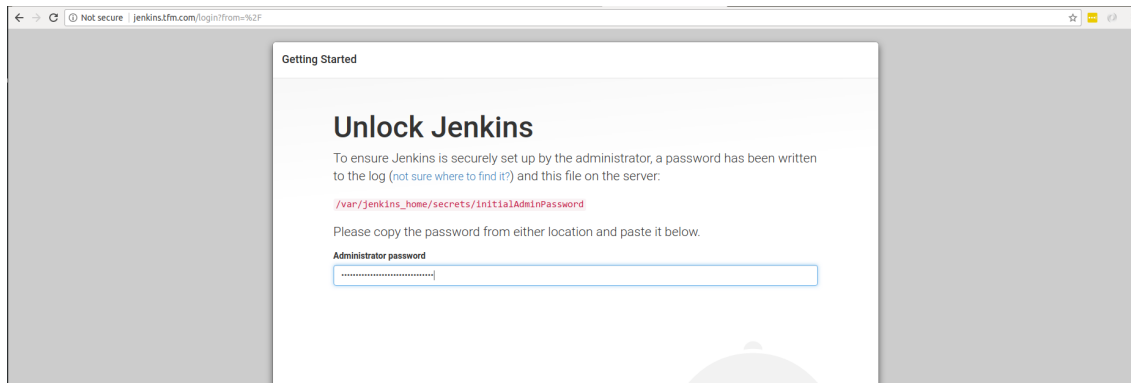


Figura 4.4 Bienvenida a la aplicación.

```

eleazarr [~] $ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
entorno_ruby-tool    latest             e696be922b90       About a minute ago 118MB
entorno_jenkins      latest             dceb43b3976d       About a minute ago 844MB
entorno_nodejs-tool  latest             0d972eb29a61       2 minutes ago      60.7MB
entorno_clairctl     latest             7dce6682cbff       3 minutes ago      1.37GB
quay.io/coreos/clair-git latest             02c150a7eb41       14 hours ago       422MB
postgres             latest             eb0f0e9af6d2       14 hours ago       266MB
jenkins/jenkins      lts                aca6340878dd       38 hours ago       840MB
golang               latest             5e2f23f821ca       2 days ago         727MB
ruby                 2.1.10-alpine     614872f94ade       2 months ago       118MB
node                 6.10.2-alpine     24d2680547f5       4 months ago       54.1MB
eleazarr [~] $ docker exec -ti jenkins bash
jenkins@4f58433cdacf:/ $ cat /var/jenkins_home/secrets/initialAdminPassword
4c23c9dad3c74e52b869ef223e635c8c
jenkins@4f58433cdacf:/ $

```

Figura 4.5 Encontrando el código oculto.

Una vez superado la pantalla de bienvenida a la aplicación, se podrá decidir los plugins que serán instalados al inicio (Figura 4.6) y comenzará el proceso de instalación inicial de plugins (Figura 4.7).

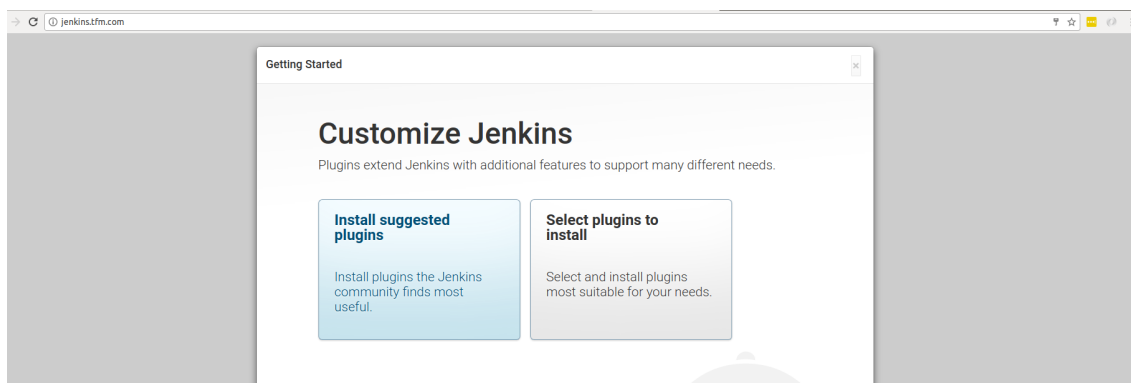


Figura 4.6 Instalando plugins recomendados.

El último paso de la preparación inicial será crear al primero de los usuarios administrador de la aplicación, la Figura 4.8 muestra al usuario creado.

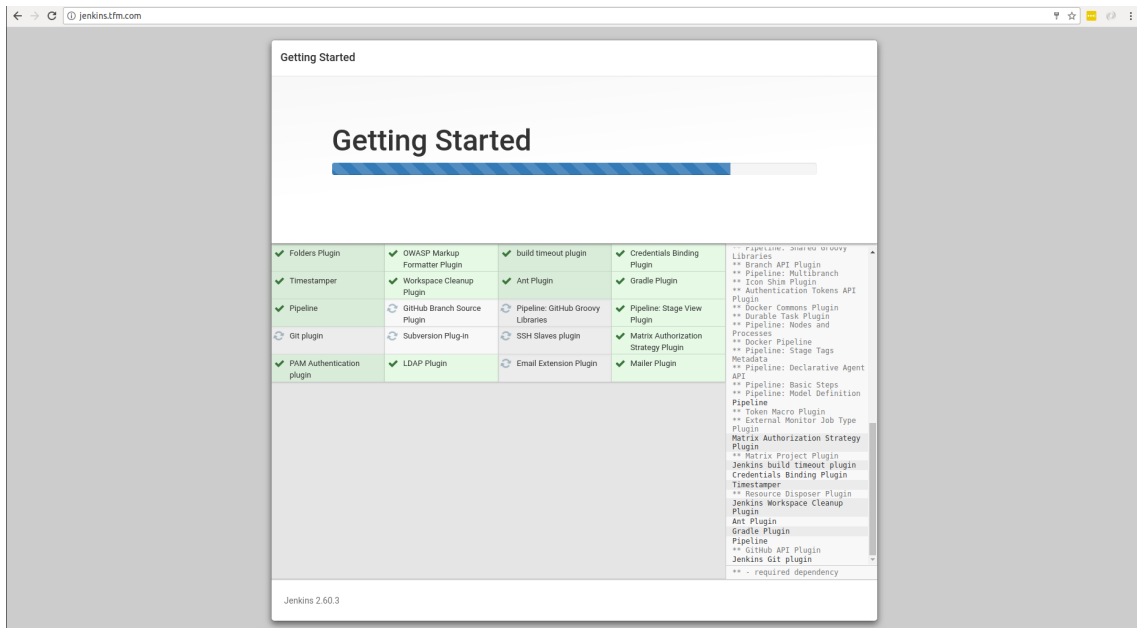


Figura 4.7 Proceso de instalación de plugins.

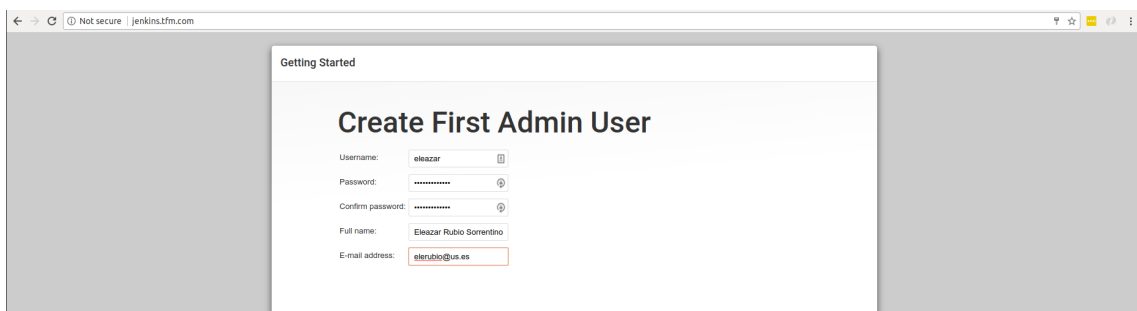


Figura 4.8 Creando el primer usuario administrador.

Con esto la aplicación queda configurada y puede comenzar a usarse (Figura 4.9).

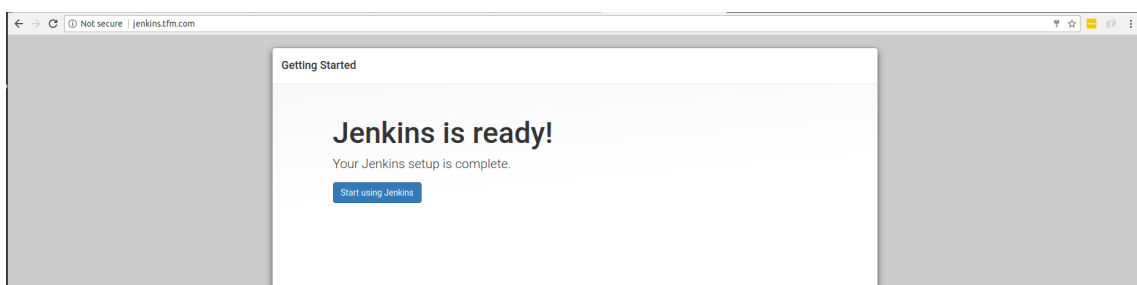


Figura 4.9 ¡Jenkins está listo!.

Una vez Jenkins está configurado, lo primero que se deberá hacer para poder llevar a cabo el objetivo del presente TFM será instalar el plugin *HTML_publisher*, puesto que será utilizando este como se crearán los informes HTML resultado de la ejecución de cada análisis estático. Las figuras 4.10 y 4.11 muestran las pantallas recorridas.

Tras aplicar la instalación del plugin, y como muestra la Figura 4.12, Jenkins se reiniciará, devolviendo

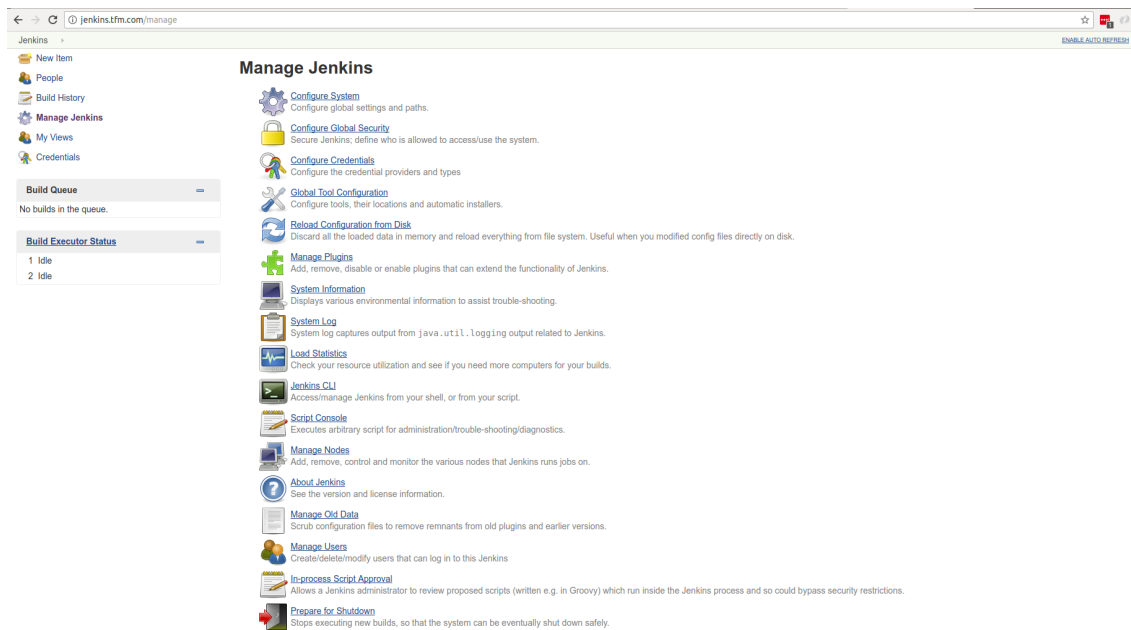


Figura 4.10 Manage Jenkins -> Manage Plugins.

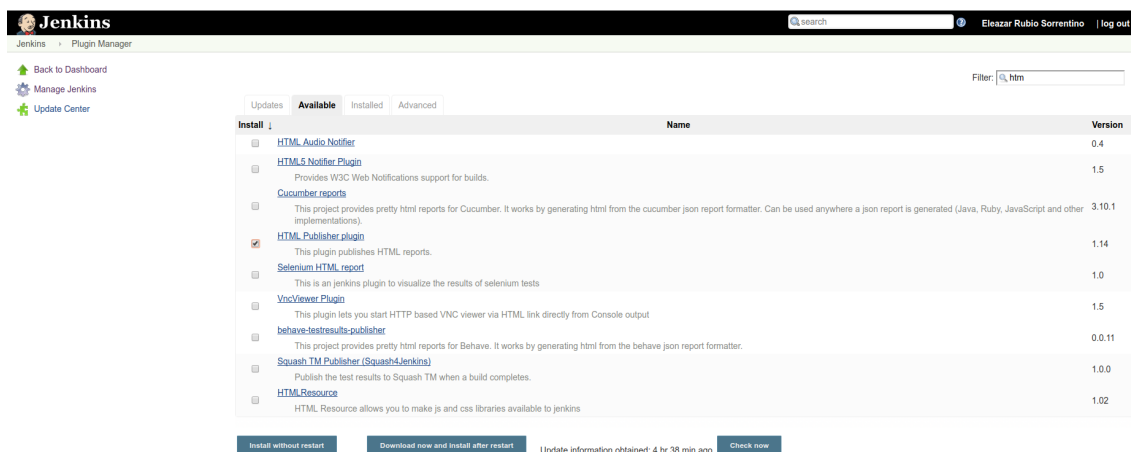


Figura 4.11 Instalando HTML_publisher.

al usuario al panel principal (Figura 4.13) de la aplicación, con el entorno completamente configurado para comenzar a añadir los trabajos de Jenkins que automatizarán el proceso de análisis.

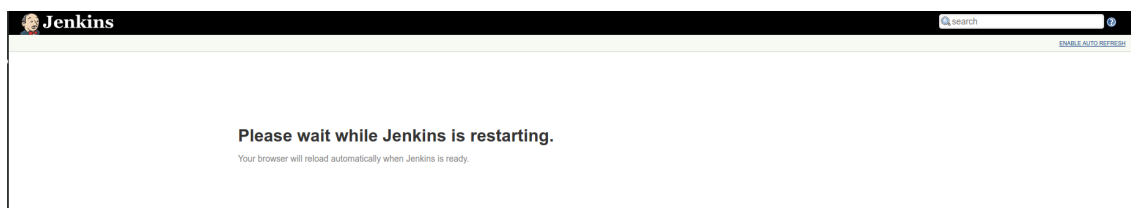


Figura 4.12 Reiniciando la aplicación tras la instalación del plugin.

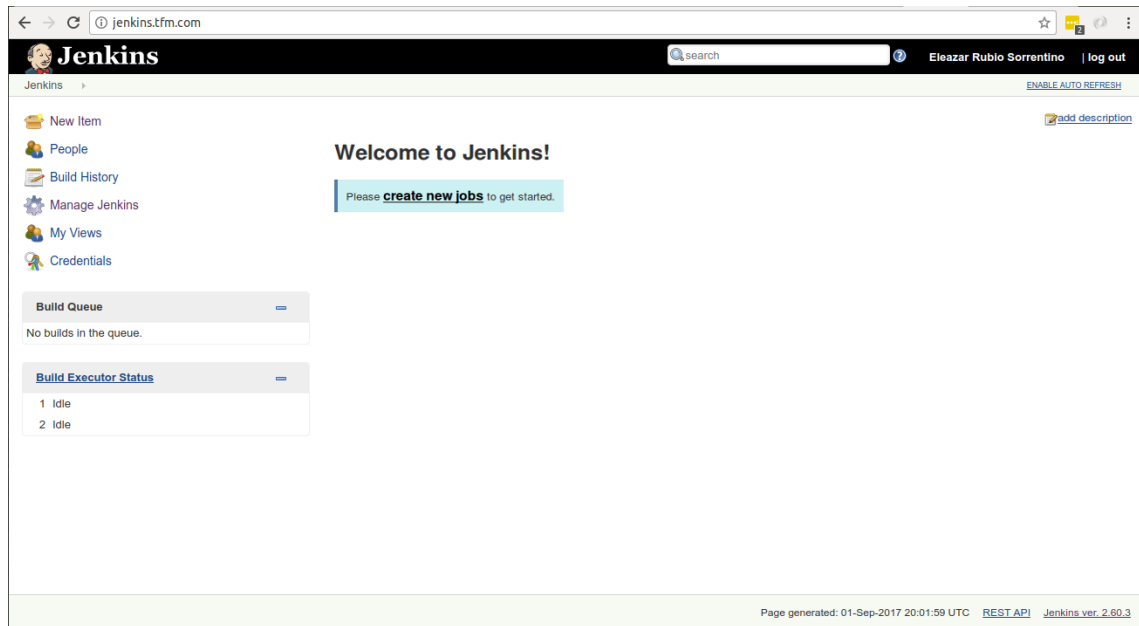


Figura 4.13 Pantalla de inicio de Jenkins CI.

4.2 Trabajando con Jenkins

En la sección actual se va a utilizar el contenido empleado en los apartados de la Sección 4.1 para mostrar al lector la creación de tareas o trabajos en la plataforma de IC Jenkins, con el fin de automatizar el proceso de análisis de las dependencias de código de aplicaciones escritas mediante los lenguajes de programación Ruby y Node.js, así como el análisis de las posibles vulnerabilidades que pudieran mantener los contenedores de Docker utilizados por la compañía.

4.2.1 Análisis de imágenes Docker

4.2.2 Análisis de dependencias Ruby

4.2.3 Análisis de de dependencias Node.js

4.3 Evaluación de la solución

ALGUNAS EJECUCIONES QUE MUESTREN VULNERABILIDADES Y SOLUCIONAR ALGUNAS.

4.4 Resultados obtenidos

CIERRE DEL APARTADO

5 Conclusiones

Security is not a line in the sand. Protecting your business, customers, citizens' data, should be always your number one priority

DR. WERNER VOGELS, 2017
CTO AT AMAZON.COM

El apartado actual, tras haber recorrido en la memoria de este TFM "Seguridad en la integración continua de la metodología ágil y la filosofía DevOps" toda la información relativa al proyecto, está dedicado a presentar los objetivos alcanzados y a presentar las líneas futuras de trabajo en torno al mismo.

El conjunto de aplicaciones aportados a las herramientas y entorno utilizados surge ante la necesidad de las empresa moderna de vigilar de una manera no intrusiva los posibles agujeros y vulnerabilidades de seguridad que contengan las aplicaciones desarrolladas por ellas y los sistemas donde son desplegados. Esta tarea viene requiriendo un esfuerzo excesivo de parte de aquellas personas que se preocupan por mantener los sistemas lo más protegidos posible, aún cuando el ... puede estar originado en la multitud de dependencias que puede requerir una aplicación de gran envergadura o la inmensa cantidad de imágenes de diferente naturaleza que se despliegan en forma de contenedor en los entornos que dan soporte a las aplicaciones que sustentan el negocio.

Siguiendo esta ... se decide aportar al proceso de Integración Continua de empresas Tecnología de la información (TI) un sencillo mecanismo de despliegue de contenedores que contendrán un conjunto de herramientas y utilidades con las que poder realizar tareas de análisis estático de manera pasiva como parte del proceso de trabajo natural de la compañía, analizando... , advirtiendo en caso de ... aportando

Por este motivo se puede concluir que ... herramientas aportadas... junto con los trabajos automatizados... y ... cumple en conclusión con el objetivo perseguido en la realización de este TFM.

Sin embargo, como es trivial en todo proceso tecnológico, la solución aquí entregada no es estática e inamovible y se presta a ser mejorada, además de a mantener los resultados obtenidos actualizados en el tiempo.

Muchas de las tareas y líneas de avance que este nuevo reto contempla aún son desconocidas, ya que van a surgir en la utilización por parte de los usuarios de la solución aportada. Otras, en cambio, pueden ser establecidas desde el momento actual. A continuación se presentan algunas de las tareas que conforman la línea futura de trabajo de esta aplicación, identificadas durante la realización del proyecto y la redacción de esta memoria:

- Aumentar el número de lenguajes de programación contemplados en los análisis estáticos: existen

múltiples herramientas en la actualidad de características similares a las aquí presentadas que cubren un amplio abanico de lenguajes de programación diferentes. Implementar un conjunto mayor de dichas herramientas como parte del conjunto actual es una notable mejora a lo aquí presentado.

- Sistema de notificaciones del resultado de los análisis:
- Preparación del entorno para el despliegue en producción:
- Ampliar los repositorios alcanzados:

Alcanzado este punto se puede concluir que, con este Trabajo Fin de Máster TFM, se abarca un aspecto importante ... seguridad en las tecnologías de la información... y la tarea diaria de un experto en ...: detección, estudio y solución a un problema de seguridad determinado, mediante la aplicación de la tecnología y los recursos disponibles.

Además, el resultado obtenido es un producto con utilidad tangible, que ya está siendo utilizado en los sistemas de Workshare Inc., proporcionando valor añadido a la actividad en materias de seguridad ... que allí se realiza y permitiendo Este aspecto es elemento clave a la hora de diferenciar el proyecto realizado, ya que existen múltiples trabajos realizados que nunca llegan a superar la barrera de estudios teóricos, para convertirse en un producto o herramienta final en completo funcionamiento, lo que sin duda es una enorme satisfacción para su autor.

Apéndice A

Repositorio de GitHub

Share your knowledge. It's a way to achieve immortality.

DALAI LAMA
HIGH LAMA OF TIBETAN BUDDHISM

Todo el contenido de la memoria actual para el Trabajo fin de Máster TFM "Seguridad en la integración continua de la metodología ágil y la filosofía DevOps", así como los archivos y la información necesaria para desplegar el entorno de pruebas realizado se encuentran a disposición pública en los repositorios de Github, con el único objetivo de seguir avanzando en el desarrollo de la idea y que permanezca disponible a cualquier usuario que pudiera beneficiarse de la misma. La Figura A.1 muestra el repositorio comentado.

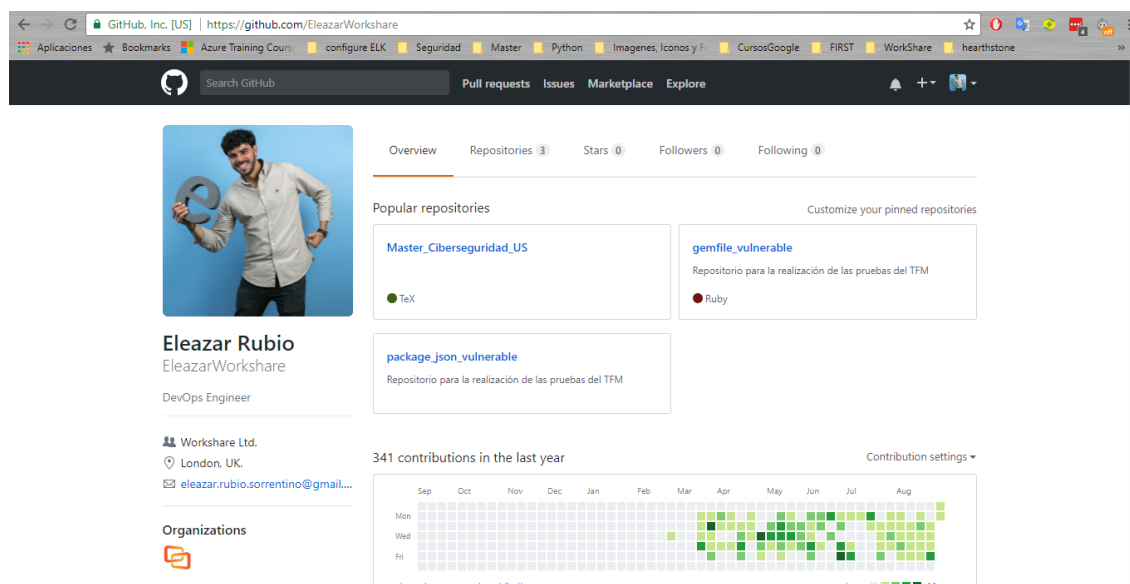


Figura A.1 <https://github.com/EleazarWorkshare>.

Por supuesto, cualquier contribución será bien recibida.

Índice de Figuras

1.1	Introducción al proceso DevOps	2
2.1	CAMBIAR LA IMAGEN	6
3.1	Logotipo de GitHub	10
3.2	Escaneo de vulnerabilidades en las dependencias de Node JS	12
3.3	Logotipo de Docker	13
3.4	Pila de funcionamiento de docker	13
3.5	Clair	14
3.6	Esquema de escaneo y detección de vulnerabilidades en imágenes Docker	15
3.7	IC con Jenkins	15
3.8	Slack	17
4.1	Despliegue del entorno con Docker Compose	22
4.2	Comprobando la utilidad nsp	24
4.3	Informe generado por Clairctl	26
4.4	Bienvenida a la aplicación	27
4.5	Encontrando el código oculto	27
4.6	Instalando plugins recomendados	27
4.7	Proceso de instalación de plugins	28
4.8	Creando el primer usuario administrador	28
4.9	!Jenkins está listo!	28
4.10	Manage Jenkins -> Manage Plugins	29
4.11	Instalando HTML_publisher	29
4.12	Reiniciando la aplicación tras la instalación del plugin	29
4.13	Pantalla de inicio de Jenkins CI	30
A.1	https://github.com/EleazarWorkshare	33

Índice de Códigos

3.1	Ejemplo de uso de bundler-audit	11
3.2	Ejecutando bundler-audit tras la actualización de las vulnerabilidades conocidas	12
3.3	Ignorar vulnerabilidades con bundler-audit	12
4.1	Definiendo Clair en Docker Compose	20
4.2	Definiendo PostgreSQL en Docker Compose	21
4.3	Definiendo Jenkins en Docker Compose	21
4.4	Definiendo Las herramientas de análisis estático en Docker Compose	21
4.5	Comando de ayuda de docker exec	22
4.6	Comando de ayuda de docker exec	23
4.7	Probando bundle-audit desde el contenedor	23
4.8	Generando un informe HTML con Clairctl	25

Bibliografía

- [1] Israel Alcázar, *Introducción a Git y Github*, Junio 2014, [Enlace](#).
- [2] Elena Arrieta, *DevOps: la tecnología y el negocio deben hablar el mismo idioma*, Mayo 2017, [Enlace](#).
- [3] Claranet, *DevOps: qué es y cómo lo aplicamos*, Accedido: Agosto de 2017, [Enlace](#).
- [4] coreos/clair, Accedido: Septiembre de 2017, [Enlace](#).
- [5] Ana M. del Carmen García Oterino, *¿Qué es Docker? ¿Para qué se utiliza? Explicado de forma sencilla*, Julio 2015, [Enlace](#).
- [6] Alan R. Earls, *Construir un entorno DevOps con microservicios y contenedores*, Diciembre 2015, [Enlace](#).
- [7] Hewlett Packard Enterprise, *Application Security and DevOps*, Octubre 2016, [Enlace](#).
- [8] Docker Inc., Accedido: Septiembre de 2017, [Enlace](#).
- [9] ———, *Install Docker*, Accedido: Septiembre de 2017, [Enlace](#).
- [10] ———, *Install Docker Compose*, Accedido: Septiembre de 2017, [Enlace](#).
- [11] GitHub Inc., Accedido: Agosto de 2017, [Enlace](#).
- [12] Consultor IT, *Estudio CA Technologies sobre la importancia de la Agilidad y DevOps en el desarrollo de software [pdf de 20 pgs.]*, Enero 2017, [Enlace](#).
- [13] Jenkins, Accedido: Agosto de 2017, [Enlace](#).
- [14] Adam Prescott, *Automated vulnerability checking with bundler-audit and Travis*, Junio 2015, [Enlace](#).
- [15] Erlinis Quintana, *10 razones para usar Github*, Junio 2015, [Enlace](#).
- [16] Margaret Rouse, *static analysis (static code analysis)*, Accedido: Septiembre de 2017, [Enlace](#).
- [17] rubysec/bundler audit, Accedido: Septiembre de 2017, [Enlace](#).
- [18] rubysec/ruby-advisory db, Accedido: Septiembre de 2017, [Enlace](#).

- [19] Pedro Santamaría, *Slack, aprende a sacarle partido a la herramienta de comunicación de moda*, Abril 2015, [Enlace](#).
- [20] CA Technologies, *Accelerating Velocity and Customer Value with Agile and DevOps*, Enero 2017, [Enlace](#).
- [21] Rubén Velasco, *¿Qué es Docker? ¿Para qué se utiliza? Explicado de forma sencilla*, Febrero 2016, [Enlace](#).
- [22] Macarena Vera, *Los beneficios de implementar la Metodología Ágil*, Accedido: Septiembre de 2017, [Enlace](#).
- [23] Thought Works, *Continuous Integration*, Accedido: Septiembre de 2017, [Enlace](#).

Glosario

CI	Continuous Integration III, V
DevOps	Development and Operations III, V, 1, 2
OS	Operative System V
QA	Quality Assurance III, V
SaaS	software como servicio III, V
SO	Sistema Operativo III
SW	software III, V
TFM	Trabajo Fin de Máster III, 1–3