Universidad Internacional de La Rioja

Escuela Superior de Ingeniería y Tecnología

Máster Universitario en Desarrollo y Operaciones (DevOps)

Implementación de la infraestructura y medidas de seguridad a nivel de contenedores para el despliegue de una aplicación web para la gestión de conocimientos aplicando un ciclo DevOps en un entorno de nube pública

Trabajo fin de estudio presentado por:	Pedro Santino Suntaxi Cocanguilla
Tipo de trabajo:	TFM
Director/a:	Juan Manuel González Calleros
Fecha:	20 de julio de 2022

Resumen

Se desarrolló una aplicación web progresiva que se utiliza como base de conocimientos para un departamento de una pequeña empresa (Un software para base de conocimiento es un programa que ayuda a una empresa a centralizar, almacenar, organizar y compartir información), y por motivos de seguridad se encuentra desplegada en un servidor sin salida a internet. El acceso a la aplicación solo es posible conectándose a una red privada de la compañía. Existe la necesidad que sea accesible desde cualquier lugar del mundo y se propone migrar toda la aplicación a una nube pública AWS aplicando un ciclo DevOps con Jenkins. La aplicación utilizará tecnología de contenedores como Docker y para el despliegue automatizado de la infraestructura tecnologías como Terraform y Ansible. La aplicación contiene información sensible y debe estar disponible el mayor tiempo posible, por lo cual el tema de seguridades de red y del contenedor es muy importante. Se tomará como guía a OWASP, documentos académicos o libros para implementar medidas preventivas en temas relacionados con la seguridad a nivel de contenedores. Como resultado final del presente trabajo la aplicación se desplegará de forma automática aplicando DevOps y teniendo un énfasis en la seguridad.

Palabras clave: DevOps**,** Contenedores**,** Seguridad informática**,** Terraform**,** Ansible

Abstract

A progressive web application was developed to be used as a knowledge base for a department of a small company (A knowledge base software is a program that helps a company to centralize, store, organize and share information), and for security reasons it is deployed on a server without internet access. Access to the application is only possible by connecting to the company's private network. There is a need to make it accessible from anywhere in the world and it is proposed to migrate the entire application to an AWS public cloud applying a DevOps cycle with Jenkins. The application will use container technology such as Docker and for automated deployment of the infrastructure technologies such as Terraform and Ansible. The application contains sensitive information and must be available as long as possible, so the issue of network and container security is very important. OWASP, academic papers or books will be taken as a guide to implement preventive measures on issues related to container level security. As a final result of this work the application will be deployed automatically applying DevOps and having an emphasis on security.

Keywords: DevOps, Containers, IT Security, Terraform, Ansible

Índice de contenidos

Índice de figuras

Índice de tablas

1. Introducción

"DevOps no es una herramienta o tecnología, es un enfoque o cultura que mejora las cosas". (Soni, 2016, pág. 8), y una de sus mayores ventajas es la de facilitar el trabajo en conjunto y colaborativo entre los roles de operaciones y desarrollo, que por lo general trabajan de forma separada. Por tal razón, en el presente trabajo se utiliza el enfoque DevOps para migrar una aplicación web progresiva utilizada para la gestión de conocimientos a una nube pública con el proveedor Amazon Web Services AWS.

Para cumplir con el objetivo planteado es necesario utilizar herramientas o tecnologías que faciliten la construcción del pipeline y proporcionen mayor agilidad en el desarrollo. De este modo, "Muchas organizaciones están ejecutando aplicaciones en entornos nativos de la nube, utilizando contenedores y orquestación para facilitar la escalabilidad y la resiliencia." (Rice, 2020, pág. 9).

Se propone implementar el ciclo de integración y entrega continua mediante Jenkins manejando contenedores ya que como también lo menciona (Agarwal, 2021, pág. 4). "Los contenedores están de moda y por una excelente razón. Resuelven el problema más crítico de la arquitectura informática: ejecutar software fiable y distribuido con una escalabilidad casi infinita en cualquier entorno informático."

No obstante, hay que considerar que la aplicación estará expuesta al mundo y hoy en día los ataques informáticos han ido evolucionando y al formar parte de un equipo DevOps podemos plantearnos las siguientes incógnitas ¿Cómo se realizan despliegues de contenedores de forma segura? Y ¿Cómo se puede mejorar la seguridad en los contenedores?

Para poder despejar dichas dudas se utilizará como ayuda a OWASP y su top 10 de vulnerabilidades detectadas en el tema de contenedores, además de libros o artículos académicos, lo que permitirá investigar y aplicar medidas preventivas necesarias al momento del despliegue mediante un ciclo de DevOps, y evitar caer en malas prácticas que pueden dejar expuestas las aplicaciones desplegadas.

Como resultado final la aplicación funcionará en un entorno de nube pública empleando un ciclo completo DevOps con integración y entrega continúa manteniendo un énfasis en la seguridad de los contenedores, y de este modo mitigar posibles ataques.

1.1. Justificación del trabajo

Una pequeña empresa tiene un equipo de desarrollo que entrega soluciones de software adaptadas según los requerimientos de la misma. Por tal motivo, se solicita migrar una aplicación PWA utilizada como base de conocimientos a un entorno de nube pública. Y para cumplir con este objetivo el equipo de desarrollo desea implementar la filosofía DevOps, ya que su forma de trabajo actual se basa en un modelo de cascada por lo cual migrar una aplicación a un entorno de nube es complejo.

No obstante, también en necesario mitigar el riesgo de exposición ante ataques informáticos, ya que la aplicación web contiene información sensible y como antecedente la empresa sufrió múltiples ataques informáticos. Por lo tanto, se propone utilizar el Marco de trabajo DevOps para implementar un ciclo completo e investigar sobre las medidas necesarias a tomar para proteger la aplicación poniendo énfasis a los contenedores, ya que en un entorno de pruebas o educativo muchas veces no se toma en cuenta estos aspectos, es necesario realizar el presente trabajo.

1.1.1. Problema a tratar

La aplicación PWA se encuentra alojada en un servidor propio de la empresa, que solo es accesible para la red privada de la misma. Y el objetivo es automatizar el ciclo de integración y entrega para migrarla a un entorno de nube pública utilizando una filosofía de trabajo DevOps.

La aplicación estará alojada en una nube pública accesible desde cualquier lugar del mundo y utilizará una tecnología de contenedores llamada Docker. Por lo tanto, es necesario investigar sobre los riesgos de seguridad que pueden ocurrir, enumerar las posibles amenazas, priorizar su grado y definir un enfoque para su mitigación.

1.1.2. Causas del problema

Al ser un marco de trabajo nuevo, en la empresa no existe un procedimiento a seguir para migrar una aplicación ni para implementar un pipeline DevOps.

El conocimiento y la experiencia adquirida en sistemas tradicionales basados en servidores o sistemas basados en máquinas virtuales se deben adaptar a los despliegues basados en contenedores

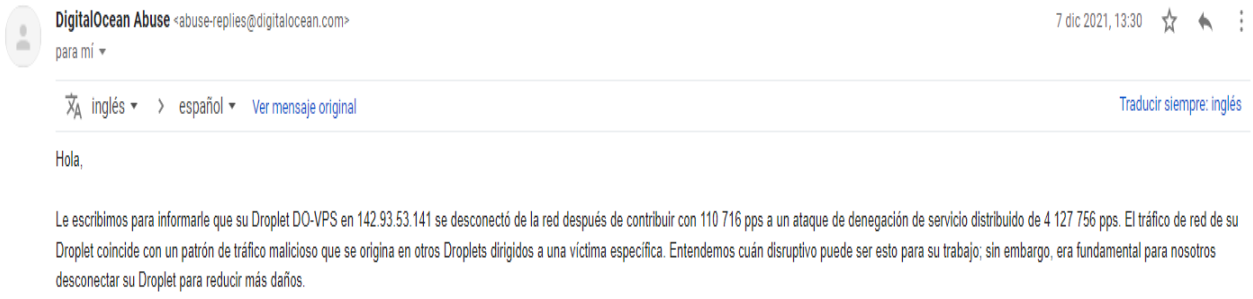
Utilizar una tecnología como Docker facilita el trabajo para la entrega del producto, pero así mismo se puede tener fallas de seguridad ya sea por descuido en su configuración, o por no seguir buenas prácticas en su generación.

1.1.3. Relevancia del problema

Es relevante porque en el año 2021 se probó por primera vez la tecnología Docker para desplegar una aplicación de prueba, pero el contenedor que funcionaba en el proveedor Digital Ocean sufrió un ataque informático como se muestra en la Figura 1, convirtiéndose en un contenedor Zombie (Contenedor infectado por un virus informático y utilizado para realizar labores maliciosas remotamente). Por lo tanto, es necesario mitigar los fallos de seguridad para evitar sanciones o bloqueos que pueden perjudicar judicialmente o económicamente a la empresa.

El problema es relevante porque se investigará sobre las medidas a implementar para proteger un contenedor de aplicaciones con Docker, y que servirá como un modelo para el despliegue de cualquier aplicación en una nube pública utilizando la filosofía DevOps.

Figura 1. *Notificación de correo sobre una falla de seguridad por un contenedor zombie montado en un Droplet de Digital Ocean*



(Fuente: Autoría Propia)

1.2. Planteamiento del problema

El presente trabajo pretende dar una solución al requerimiento de una pequeña empresa para realizar la migración de una aplicación Web a la nube, ya que por su crecimiento es necesario utilizar nuevas tecnologías que ayuden a potenciar el normal desarrollo de sus actividades y dejar de utilizar infraestructura propia, que por motivos de antigüedad y costos no es viable su actualización.

Pero utilizar un entorno de nube y un conjunto de herramientas que facilitan la construcción de un ciclo de vida utilizando la filosofía DevOps también trae consigo nuevos retos para su implementación e inquietudes por parte de la empresa, en especial a los temas relacionados con la seguridad de sus aplicaciones.

Ya que como experiencia pasada en la empresa, al probar una herramienta de contenedores llamada Docker y tener un pequeño fallo en su configuración, se produjo un serio daño en su infraestructura.

Pero existen muchas más ventajas al aplicar el marco de trabajo DevOps tales como implementar procesos de automatización que soporten integración y entrega continua, desarrollar código de forma ágil, testearlo y desplegarlo en conjunto con su infraestructura. Sin embargo, para un entorno de producción es necesario implementar medidas de seguridad a nivel de red y en las herramientas utilizadas, en este caso se utilizará una tecnología como Docker y es necesario investigar e implementar seguridades a nivel de contenedores para evitar amenazas o reducir el riesgo de que se produzcan.

Al contar con un equipo de trabajo pequeño es complejo abarcar todo el tema de seguridades en sus diferentes fases, sean estas antes o después de la entrega del producto, ya que el presente trabajo también busca migrar una aplicación existente e implementar un proceso de automatización del ciclo de vida con el marco de trabajo DevOps, de este modo se tendrá un énfasis en el tema de seguridades en los contenedores y mediante un proceso de investigación tener todo el conocimiento necesario para realizar ciertas configuraciones que ayuden a mejorar la seguridad y evitar la implementación de malas prácticas, lo que desencadenaría en riesgos de vulneración de las aplicaciones que pueden representar gastos económicos o problemas legales.

1.2.1. Solución del problema

Una posible solución es utilizar imágenes de contenedores Docker libres y que están alojadas en Docker Hub e integrarlas al pipeline CI/CD, pero no cubriría todas las necesidades particulares que se necesita, además no hay la certeza sobre la confianza en la generación de dichas imágenes, ya que se puede alojar alguna vulnerabilidad o código malicioso.

Como solución a este problema se propone realizar el flujo completo de DevOps (pipeline CI/CD) y construir los contenedores. De este modo, se tiene un control personalizado y se puede profundizar en la investigación sobre los riesgos de seguridad actuales, entregando una contribución y valor agregado a la automatización del ciclo de vida de la aplicación.

Todo el código y herramientas utilizadas estarán disponibles al público en general para que se pueda mejorar y agregar nuevas características.

1.2.2. Propuesta de la solución

Se propone como primer paso implementar todo el ciclo automatizado de integración y entrega continua para la aplicación en el proveedor de nube Amazon Web Services, utilizar la herramienta de código abierto Jenkins para la automatización y utilizar un sistema de control de versiones como GitHub.

Para construir la infraestructura de red en la nube pública se propone utilizar la herramienta de IaC Terraform y como tecnología de contenedores a Docker.

Una vez el ecosistema este construido y el ciclo de vida de la aplicación este automatizado, se procederá a la implementación de las medidas de seguridad a nivel de los contenedores tomando en cuenta los siguientes escenarios según (Rice, 2020, pág. 3):

- Atacantes externos pueden realizar intentos para acceder desde el exterior a un despliegue

- Atacantes internos pueden tener acceso parcial o completo al despliegue

- Actores internos tales como desarrolladores o administradores tienen malas intenciones y pueden obtener algún nivel de privilegio y conseguir acceder al despliegue

- Actores internos de forma involuntaria o accidental logran causar problemas

También es importante analizar que cada actor tiene un conjunto determinado de permisos que hay que tomar en cuenta formulándose las siguientes incógnitas:

- ¿A qué recursos se tiene acceso con las credenciales asignadas?

- ¿Tienen acceso a las cuentas de usuario de los equipos anfitriones en los que se ejecuta la implementación?

- ¿Qué permisos tienen en el sistema y como es su acceso a la red?

- ¿Qué partes del sistema pertenecen a la nube privada virtual (VPC)?

Como resultado final se automatizará el ciclo de vida de la aplicación en la nube pública y se implementará las respectivas medidas de protección para evitar o mitigar amenazas en los contenedores.

1.3. Estructura de la memoria

El presente TFM está conformado por 7 capítulos distribuidos como se muestra a continuación:

- **Capítulo 1. Introducción.** En este apartado se describe el contenido del trabajo, la motivación y todos los objetivos a cumplir.
- **Capítulo 2. Estado del arte**. Se realiza una búsqueda de bibliografía, trabajos o soluciones existentes que cubran el tema del presente TFM para realizar una comparación y añadir una contribución en la problemática planteada.
- **Capítulo 3. Objetivos concretos y metodología de trabajo**. Se definen los objetivos generales y específicos del TFM, así como el proceso a realizar para alcanzarlos.
- **Capítulo 4. Implementación de la solución**. Se describe todo el proceso para alcanzar la solución propuesta y como resultado final obtener todo el sistema funcionando en la nube.
- **Capítulo 5. Conclusiones y trabajo futuro**. Se especifica los resultados obtenidos a lo largo del desarrollo de la solución y se plantea las conclusiones del trabajo
  Se define las líneas de investigación para mejorar el trabajo a futuro.
- **Capítulo 6. Referencia Bibliográfica.** Se muestra las fuentes consultadas, libros, artículos científicos, etc. que ayudaron en la elaboración del TFM

- **Capítulo 7**. **Anexos.** Contiene todos los documentos, imágenes, tablas que respalden el desarrollo del presente TFM.

2. Contexto y estado del arte

En el presente capítulo se describe el contexto del trabajo, y se realiza una investigación en base a referencias de trabajos similares en diferentes fuentes bibliográficas. Así, también se define y describen temas importantes que ayudan a comprender la temática del estudio y como resultado final se encuentra el estado del arte que contiene los trabajos relacionados que plantean soluciones para la construcción del sistema de integración y entrega continua con énfasis en la seguridad a nivel de contenedores utilizando la filosofía DevOps.

2.1.Contextualización y antecedentes

2.1.1. DevOps y sus herramientas

DevOps reúne una serie de principios y prácticas que representa una filosofía. De este modo, fomenta la colaboración mutua entre los equipos de desarrollo y operaciones durante todo el ciclo de vida en el desarrollo de software, su mantenimiento y las operaciones.

Las organizaciones que buscan implementar y poner en práctica DevOps utilizan una serie de procesos y herramientas con el objetivo de automatizar todo el ciclo de entrega de software y hacerlo de forma ágil a través de un proceso de integración y entrega continua (CI/CD), y a su vez supervisar las aplicaciones en producción (Agarwal, 2021).

Debido a la pandemia muchas organizaciones se vieron forzadas a cambiar su forma de trabajo y buscar nuevas alternativas ante la necesidad de actualizar las aplicaciones existentes, todo esto alineado a un proceso de transformación digital.

Esto representa un reto para los equipos de desarrollo de software y TI en las empresas, y se genera la incógnita sobre cuál es la mejor manera para migrar una o más aplicaciones a un entorno de nube.

No existe un estándar sobre las herramientas de apoyo, aplicaciones, procesos y tecnologías que son necesarios para alcanzar dicho objetivo

No obstante, en los últimos años han surgido herramientas que facilitan este objetivo entre las cuales encontramos Docker, Jenkins, Terraform, Github, etc.

2.1.2. Beneficios de la tecnología de contenedores en el ciclo de vida de una aplicación

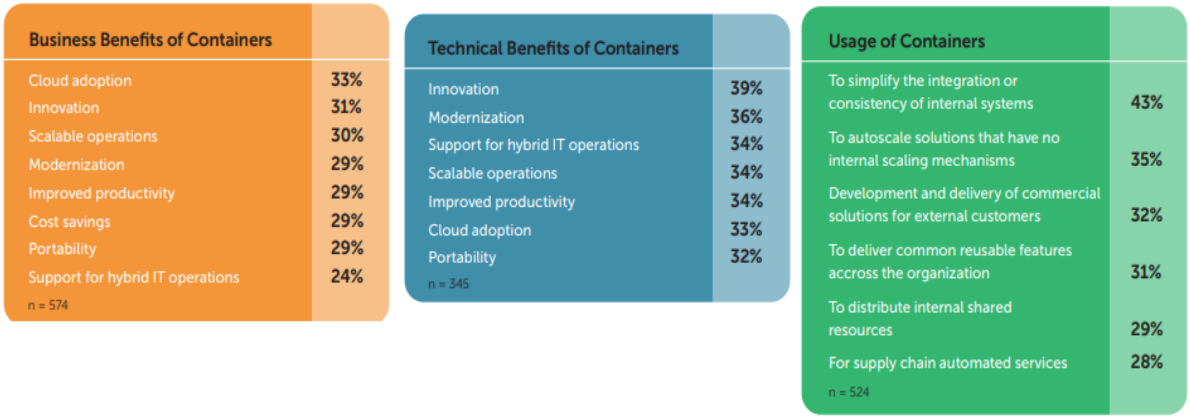
"Entre los meses de enero y marzo de 2021, CCS Insight encuestó a 574 profesionales de TI de empresas de distintos tamaños y sectores en Europa, Oriente Medio y África. La encuesta buscaba entender cómo está respondiendo el mercado a los contenedores" (Rotibi Bola, 2021, pág. 2).

Basado en los resultados que se visualizan en la Figura 2, podemos llegar a la conclusión que el manejo y utilización de la tecnología de contenedores ofrece a las empresas beneficios técnicos, así como de negocio. Pero el dato más importante es que gran parte de los encuestados considera que la adopción de la nube es el principal beneficio al utilizar contenedores. (Rotibi Bola, 2021, pág. 4)

El resultado obtenido es el esperado, en base al concepto de los contenedores como lo menciona (Soni, 2016, pág. 47) "Los contenedores utilizan la virtualización a nivel de sistema operativo, donde el kernel se comparte entre espacios de usuario aislados". Por lo tanto, brindan las mismas características de escalamiento y entrega que se asocia a los servicios que brinda una nube.

"De hecho, las aplicaciones que utilizan contenedores y se están desarrollando o desplegando en las organizaciones simplifican la integración, la coherencia entre los sistemas y componentes internos. Esto refuerza su idoneidad para los despliegues en la nube" (Rotibi Bola, 2021, pág. 4)

Figura 2. *Encuesta sobre los beneficios de utilizar contenedores en las organizaciones*



Fuente (Rotibi Bola, 2021)

2.1.3. Problemas de seguridad en entornos de nube y contenedores

Debido a la facilidad de uso y de acceso a herramientas para la automatización y al no contar muchas veces con una persona encargada de la seguridad de las aplicaciones ya sea porque el equipo de trabajo es pequeño o porque no se cuenta con la suficiente experiencia en el tema de seguridad de la red, el ecosistema desplegado en producción puede tener vulnerabilidades de red. Por lo tanto, es importante analizar diferentes escenarios y tomar las debidas precauciones al momento de desplegar todo el entorno a producción.

No solo es necesario tomar en cuenta consideraciones al momento de levantar la red, como por ejemplo configurar (en el caso de AWS) los grupos de seguridad, permitir o bloquear puertos, sino también en el caso de los contenedores al utilizar Docker se debe considerar los fallos de seguridad que se pueda tener por no configurar de forma correcta su despliegue o por no tomar precauciones al momento de escoger una imagen prediseñada o personalizada de acuerdo a nuestras necesidades.

Hay que tener claro que Docker es una iniciativa de código abierto que permite crear, probar e implementar aplicaciones de forma ágil y "empaqueta software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución" (AWS, 2022).

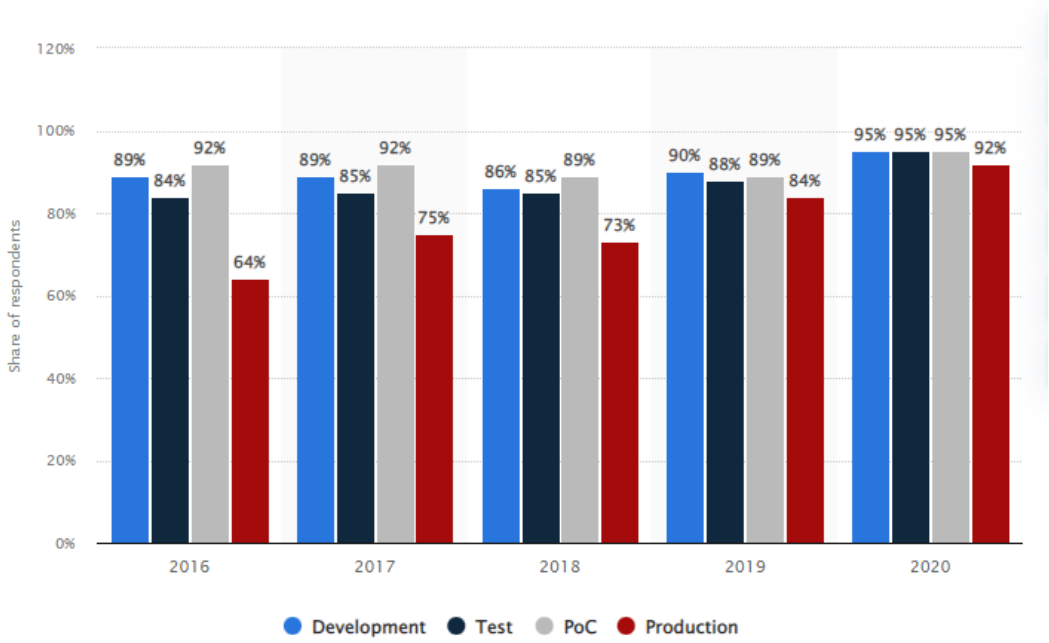
La seguridad de los contenedores está ligada con la seguridad del sistema, de la red y con un diseño arquitectónico seguro. Por lo tanto, lo mejor antes de empezar a utilizar una

tecnología de contenedores es la planificación del entorno de manera segura, puesto que realizar cambios es difícil y costoso una vez el contenedor este en producción.” (Wetter, OWASP Docker Top 10, 2020).

2.1.4. Migrar de máquinas virtuales a contenedores

En una encuesta realizada entre los años 2019 y 2020 a personas que provienen de organizaciones que se dedican a la tecnología o al desarrollo de software, muestra que ya se viene adoptando la tecnología de contenedores como se muestra en la Figura 3. Y en un principio se la utilizaba para los entornos de test, desarrollo o para realizar pruebas de concepto PoC que permiten experimentar la viabilidad técnica de una idea visualizando su funcionalidad y potencial.

Figura 3. *Uso de tecnologías de contenedores en organizaciones a nivel mundial 2016-2020*



(Sujay Vailshery, 2020)

Pero a medida que las organizaciones acumulan experiencia en el manejo de los contenedores estas tienen una tendencia a mover sus aplicaciones o sistemas a entornos de producción utilizando dicha tecnología. De este modo uno de los trabajos que realiza un ingeniero DevOps hoy en día corresponde a la tarea de migrar una o más aplicaciones que se ejecutan en máquinas virtuales a entornos de contendedor utilizando una tecnología disponible para tal efecto.

2.1.5. El ciclo de vida de DevOps basado en el término Continuo

El ciclo de vida de una aplicación basada en DevOps se divide en dos partes importantes (Soni, 2016, pág. 19):

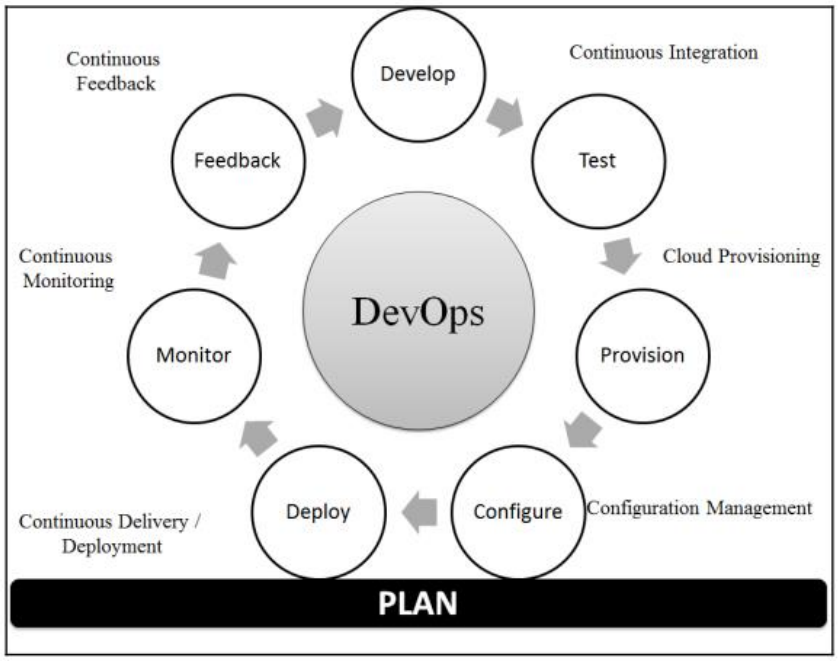
- La integración continua (CI)
- La entrega continua (CD)

La integración continua está formada por la automatización de la ejecución de las pruebas unitarias y el proceso de empaquetado.

La entrega continua se ocupa de todo el Flujo de entrega de las aplicaciones que pueden ejecutarse en diferentes entornos.

CI/CD agilita el proceso de creación de aplicaciones y automatiza fases tales como la creación, las pruebas y el análisis del código, con el objetivo de lograr la automatización de principio a fin en el ciclo de vida de la entrega de una aplicación tal como se muestra en la Figura 4. (Soni, 2016).

Figura 4. *Ciclo CI/CD en una aplicación utilizando la cultura DevOps*



(Soni, 2016, pág. 19)

2.2. Trabajos relacionados

En el presente apartado se realiza una investigación sobre las soluciones que existen en materia de automatización de despliegues con integración y despliegue continuo con énfasis en la seguridad de los contenedores.

Se analiza cada una de las herramientas que ayudan a resolver el problema planteado y se compara el presente trabajo con otros similares para validar la solución propuesta y realizar una aportación en el área de estudio.

2.2.1. Entornos CI/CD y manejo de eventos de seguridad

Aplicar la filosofía DevOps trae muchas ventajas y entre ellas la implementación de entornos CI/CD que permiten automatizar todo el ciclo de vida de una aplicación. No obstante, a pesar de las ventajas también se han generado nuevas amenazas en especial cuando se realiza un cambio cultural, y se intenta migrar de un marco de trabajo o filosofía tradicional a uno nuevo como en el caso de DevOps.

En la publicación de (Brady, Moon, Nguyen Tuan, & Coffman) "Seguridad de contenedores Docker en la computación en la nube" se describe un ecosistema de CI/CD que utiliza un analizador de imágenes Docker para detectar vulnerabilidades.

Y para medir los resultados de su solución introducen una imagen contaminada al momento de desplegar el ciclo automatizado y mediante un software analizador, se prueba las posibles vulnerabilidades que pueda poseer una imagen de Docker.

La publicación abarca el tema de la detección de vulnerabilidades a nivel de imágenes de contenedores en un ciclo CI/CD, pero sigue sin cubrir temas como la seguridad a nivel de la configuración y administración de un contenedor Docker que es el problema a resolver en el presente trabajo.

En el libro de (Soni, 2016) titulado "DevOps para el desarrollo web" describe el paso a paso para implementar un entorno CI/CD, pero no toma en cuenta la seguridad de los contenedores a pesar de que aporta valiosos conocimientos y técnicas para el despliegue de un ciclo DevOps aplicado al desarrollo web.

El texto fue lanzado en el 2016 y no cuenta no una nueva versión actualizada. Por lo tanto, muchas de las herramientas utilizadas en el libro tienen sus versiones desactualizadas y es necesario realizar una investigación para validar si existen cambios en la manera de utilizarlas en las nuevas versiones, y así aplicarlas en un ciclo CI/CD.

En el artículo publicado en la IEEE de (shah, Dubaria, & Widhalm) enumeran una serie de herramientas necesarias para implementar un ciclo CI/CD aplicando la filosofía DevOps tales como:

- GIT (herramienta para el control de código fuente)
- Ansible (herramienta para la gestión de la configuración)
- Jenkins (herramienta que actúa como un servidor de automatización)
- Vagrant (herramienta encargada del aprovisionamiento)
- Docker (Es una herramienta de contenedores)

Con todas estas herramientas se puede construir un entorno completo de Integración y Entrega Continua. Por lo tanto, el artículo brinda información valiosa que se puede acoger para la construcción del entorno propuesto.

En el artículo “Ecosistema Docker, análisis de vulnerabilidades” de (Antony, 2018) menciona que “Los contenedores proporcionan una implementación más rápida que las máquinas virtuales y un rendimiento casi nativo”. Y realizan un estudio sobre las implicaciones de seguridad al momento de utilizarlos.

El artículo cubre un problema planteado en el presente TMF que corresponde a la seguridad de los contenedores y realiza un análisis de vulnerabilidades. De Este modo, se tiene información valiosa para la implementar medidas de protección en los contenedores que se crearán en el presente trabajo. Pero no cubre uno de los objetivos que es el de implementar todo el ciclo de integración y entrega continua, y así migrar a la nube publica una aplicación diseñada y mantenida con otra filosofía o marco de trabajo diferente a DevOps. Entonces, es necesario tomar los aportes de cada uno de los trabajos similares encontrados para así elaborar el presente trabajo de final de grado y alcanzar la solución al problema propuesto.

2.3. Conclusiones del estado del arte

En los apartados anteriores se analizó el contexto sobre la importancia de implementar un ciclo CI/CD aplicando una filosofía DevOps para lanzar a producción una aplicación en la nube.

También se analizó los retos en la seguridad que pueden surgir al momento de migrar una aplicación a la nube o utilizando herramientas como contenedores.

Como resultado se confirma que existe un problema y la necesidad de tener un ciclo de vida completo con CI/CD, y que al mismo tiempo cubra aspectos básicos de la seguridad informática para evitar que las aplicaciones queden expuestas.

Al analizar el estado del arte y realizar una comparación con los trabajos relacionados con el presente TFM se llega a las siguientes conclusiones:

- Existen muchas soluciones que abarcan y brindan una respuesta al problema planteado, pero dichas soluciones están divididas en pequeños trabajos que abarcan cada una de los temas propuestos sea lo correspondiente a CI/CD o el tema de seguridades.
- Debido a que las herramientas tecnológicas utilizadas en DevOps tienen una constante actualización y muchas veces traen consigo cambios importantes, es necesario validar y probar las soluciones encontradas ya que no se tiene la certeza si su implementación este funcional con las nuevas versiones.
- En los artículos no se tiene acceso al código de la solución implementada solo brindan información teórica y al ser un trabajo práctico es necesario construir la solución para presentarla.

En base al análisis realizado, el presente trabajo de fin de master aporta una nueva solución teniendo en cuenta los siguientes puntos:

- Se unifica en un solo trabajo la implementación de un ciclo de vida de una aplicación utilizando DevOps y el tema de seguridades, con el objetivo de lanzarla a producción, pues debe ser migrada de un servidor físico a la nube pública.

- El trabajo será desplegado en la nube y estará disponible en un repositorio público para actualizarlo en el tiempo y aceptar contribuciones en su código por parte de terceras personas.
- Se investigará y analizará medidas actuales de mitigación contra amenazas de seguridad hacia los contenedores y se presentará un trabajo práctico para utilizarlos en un ciclo de vida DevOps.

3. Objetivos y metodología de trabajo

3.1. Objetivo general

Migrar a la nube una aplicación e implementar un ciclo de integración y entrega continúas adoptando medidas de seguridad a nivel de contenedores para mitigar posibles ataques cuando la aplicación se despliegue en producción.

3.2. Objetivos específicos

- Identificar las herramientas necesarias para migrar una aplicación a la nube aplicando un pipeline de integración y entrega continua utilizando como referencia toda la documentación recibida a los largo del Master.
- Establecer las tecnologías que permitan implementar el ciclo CI/CD utilizando el marco de trabajo DevOps.
- Investigar sobre las vulnerabilidades de la tecnología de contenedores apoyándose en los informes de la fundación OWASP y otros documentos o libros para evitar que se vuelva a producir un ataque que comprometa el funcionamiento de la aplicación y evitar un posible bloqueo del servicio por parte del proveedor de nube.
- Construir un diseño para el despliegue de la aplicación en la nube que permita implementar un ciclo de integración y entrega continua
- Implementar y probar el funcionamiento de todo el ciclo CI/CD de la aplicación incluyendo las configuraciones y las medidas adoptadas para mitigar posibles ataques informáticos provenientes del exterior.

3.3. Metodología del trabajo

Debido a la complejidad de la solución y la necesidad de realizar la entrega de nuevas funcionalidades en el menor tiempo posible, se elige aplicar en el desarrollo del proyecto un enfoque ágil ya que tiene relación directa con el marco de trabajo DevOps que se propone implementar para finalizar con éxito el proyecto.

Un enfoque tradicional se basa en una predicción y se tiene que cumplir con todos los requisitos, caso contrario se debe realizar nuevamente todo el proceso para definir nuevamente el proyecto, a diferencia del enfoque ágil que se basa en el cambio constante y sobre la marcha. Además, si el proyecto a desarrollar necesita cambios constantes es una ventaja utilizar un enfoque ágil ya que una de las características que propone el enfoque es dividir un proyecto de gran magnitud en pequeñas piezas o pequeños proyectos.

En el transcurso del proyecto hay la posibilidad de afrontar cambios, por lo tanto, se corre el riesgo de una parálisis del trabajo por el excesivo análisis de requisitos o que se produzca incertidumbre, siendo necesario tomar un enfoque ágil para poder dividir el proyecto en pequeñas piezas o sprints y evitar trabajar con grandes porciones u objetivos. Además, el enfoque ágil permitirá recolectar nuevas lecciones o experiencias por cada pequeña entrega por lo tanto se puede aprovechar y reutilizar los conocimientos, lo que ayuda en gran medida a disminuir los tiempos de entrega de cada objetivo propuesto.

Utilizar un enfoque ágil nos permite realizar un producto mínimo viable e ir aumentando su funcionalidad o características en el tiempo, y así se puede tener un retorno de inversión temprano al evitar sobre costes en el proyecto, ya que evitamos tener que realizar nuevamente todo el desarrollo e implementación de la solución.

El desarrollo ágil y DevOps son dos paradigmas que no son iguales, pero comparten muchas cualidades como la necesidad de acortar los tiempos de entrega del producto y adaptarse al cambio constante. De este modo, los dos están relacionados. Por lo tanto, en el presente proyecto es necesario utilizar el marco de trabajo DevOps por la necesidad de migrar y automatizar la integración y entrega continua de la aplicación con un enfoque a la seguridad. Y para garantizar todo el proceso, es necesario que los profesionales en el campo del desarrollo y operaciones trabajen en conjunto utilizando prácticas enfocadas a reducir el

tiempo entre el compromiso de cambio dentro de un sistema y el cambio a producción, garantizando al mismo tiempo la calidad.

3.3.1. Definición de la metodología o enfoque ágil más adecuado

Scrum

Se escoge la metodología Scrum que es un marco de trabajo para la gestión de proyectos software, que permite el trabajo colaborativo entre equipos y su filosofía es aprender a través de las experiencias de gestión, llevando a cabo una autogestión.

Para el presente trabajo los eventos de grupo se realizarán mediante un Daily Scrum que tiene como propósito el inspeccionar el progreso hacia el Objetivo Sprint, adaptando el Sprint Backlog según lo necesario.

Lean

Es un término usado para tener la conciencia de que todo negocio tiene que maximizar el valor que se da al cliente y a la vez tratar de reducir los desperdicios.

Para aplicar la filosofía Lean se utilizará un ciclo de mejora continua de Deming o Plan-Do-Check-Act (PDCA), para evitar despilfarros al momento de utilizar recursos en el proveedor de nube, también se priorizará la reutilización de código en las herramientas para evitar gasto de tiempo en la implementación de la solución.

Kanban

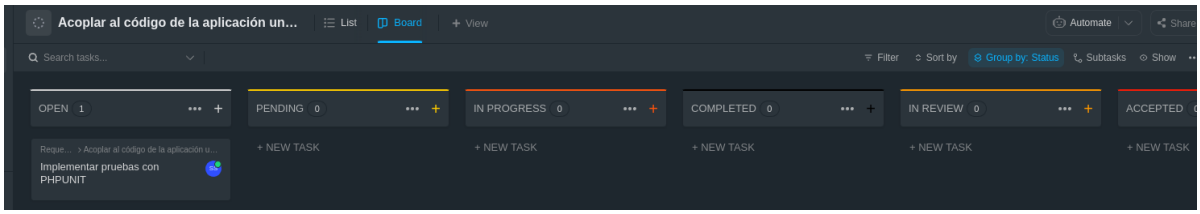
Kanban se considera tanto una metodología o conjunto de buenas prácticas como un tipo de herramienta. Y de acuerdo a los requerimientos trabajo de fin de master se propone utilizar la herramienta digital CLickUp y definir solamente 4 columnas para visualizar el trabajo que se está llevando y así agilitar todo el proceso de trabajo.

Las columnas propuestas son las siguientes:

- Abierto
- En progreso
- Revisión
- Cerrado

Tal como se muestra en la Figura 5:

Figura 5. *Tablero Kanban propuesto para el desarrollo del proyecto*



Fuente (Autoría propia)

3.3.2. Metodología de trabajo utilizada

Para el desarrollo del presente trabajo se utiliza la metodología ágil basada en Scrum y como apoyo se utiliza un tablero Kanban que ayuda a llevar el control de las historias de usuario que se van a definir a continuación mediante una lista de requisitos.

Tema: Migración de una aplicación Web a la nube utilizando el marco de trabajo DevOps con énfasis en la seguridad de los contenedores Docker

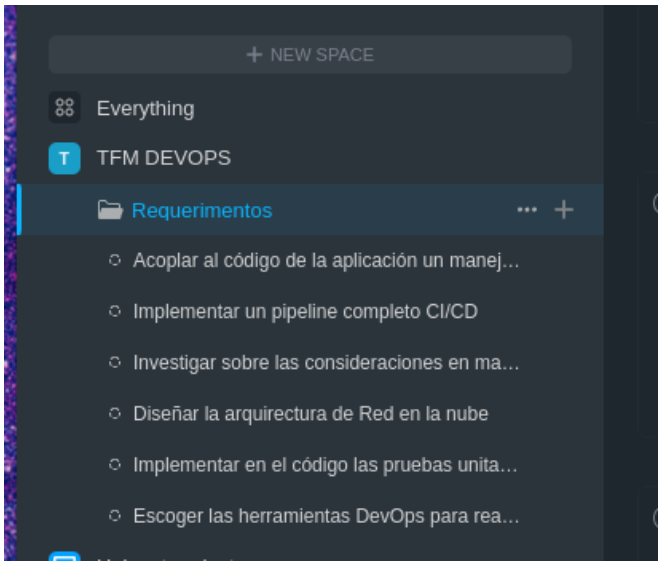
Se utiliza ClickUp como herramienta de apoyo para llevar el control del proyecto y se definen los requisitos de acuerdo a la Tabla 1, y el resultado del ingreso de los datos en la plataforma ClickUp se visualiza en la Figura 6.

Tabla 1. *Requisitos del proyecto*

	Lista de requisitos de la aplicación para implementar un ciclo CI/CD
1	Acoplar al código de la aplicación un manejador de dependencias
2	Implementar en el código las pruebas unitarias
3	Escoger las herramientas DevOps para realizar el pipeline CI/CD
4	Diseñar la arquitectura de Red en la nube
5	Investigar sobre las consideraciones en materia de seguridad de contenedores
6	Implementar un pipeline completo CI/CD

Fuente (Autoría propia)

Figura 6. *Requisitos del proyecto en la herramienta ClickUp*



Fuente (Autoría propia)

3.4. Desarrollo específico de la contribución

En el presente capitulo se detallan las tecnologías escogidas para llevar a cabo el trabajo y las ventajas que aportan. Se describe todo el proceso para el diseño y la implementación de la solución propuesta.

Como parte final se muestra su funcionamiento y se realizaran diferentes pruebas para evaluar la operatividad y el correcto funcionamiento de la misma.

3.5. Identificación de requisitos

Para realizar un ciclo CI/CD es necesario que el código de la aplicación cuente con pruebas unitarias y de esta manera implementar un ciclo de integración en el pipeline.

La aplicación web está realizada con el lenguaje de programación PHP utilizando el modelo MVC con librerías JavaScript y se detecta que no cuenta con una herramienta de manejo de dependencias.

Por tal razón se define una lista de requisitos que se deben cumplir para poder alcanzar los objetivos planteados en el presente trabajo de acuerdo a la Tabla 2.

Tabla 2. *Lista de requisitos de la aplicación*

	Lista de requisitos de la aplicación para implementar un ciclo CI/CD
1	Acoplar al código de la aplicación un manejador de dependencias
2	Implementar en el código las pruebas unitarias
3	Escoger las herramientas DevOps para realizar el pipeline CI/CD
4	Diseñar la arquitectura de Red en la nube
5	Investigar sobre las consideraciones en materia de seguridad de contenedores
6	Implementar un pipeline completo CI/CD

Fuente (Autoría propia)

3.5.1. Selección de la tecnología para el manejo de las dependencias

Para el presente trabajo se escoge la herramienta Composer que se encarga de la gestión de las dependencias para el lenguaje de programación PHP. Por lo tanto será de gran ayuda a la hora de implementar pruebas unitarias.

De acuerdo a (AcensTechnologies, 2014) entre las principales características de la herramienta están las siguientes:

- Permite escoger la versión exacta de las librerías a utilizar
- Utiliza estándares y fomenta su uso
- Cuenta con un repositorio general con una gran cantidad de librerías

3.5.2. Selección de la tecnología para implementar pruebas unitarias

Una prueba unitaria permite comprobar el resultado o salida de las funciones, clases o métodos implementados en un proyecto de programación esperando un resultado previamente configurado.

Según (Digite, 2022) los beneficios de implementar pruebas unitarias son los siguientes

- Las pruebas de regresión son mucho rápidas
- Se visualizan alertas cada vez que una prueba falla
- Los errores son más fáciles de identificar

- Facilita la implementación del ciclo de integración continua

Para el presente proyecto se utiliza la librería PhpUnit por las razones antes mencionadas.

3.5.3. Herramientas de apoyo para la implementación de un pipeline CI/CD

En el mercado existen una variedad de herramientas que son de ayuda para implementar el marco de trabajo DevOps. Y para el presente proyecto se eligen las siguientes:

3.5.4. Gestión automática de versiones del proyecto con Git y Github

(Davis & Daniels, 2016, pág. 178) Menciona que tener la capacidad de confirmar, comparar, fusionar y restaurar revisiones anteriores de código de una aplicación, fomenta una cooperación y colaboración más efectiva entre equipos. Disminuye el riesgo, porque brinda una forma de revertir los cambios realizados a versiones anteriores del código. Por lo tanto, las organizaciones deberían fomentar el uso de una herramienta de control de versiones.

Una herramienta de control de versiones es de gran ayuda para enfrentar y resolver los conflictos que se generan por tener a varias personas trabajando en un mismo archivo al mismo tiempo, y es una forma segura de realizar cambios en los ficheros y de ser necesario revertirlos. El empleo de una herramienta de control de versiones desde el inicio de un proyecto facilitará la adopción de buenos hábitos en el grupo de trabajo. (Davis & Daniels, 2016, pág. 178).

Para elegir un sistema de control de versiones (VCS) se debe considerar que la herramienta cumpla con las siguientes características:

- Permita abrir y bifurcar los repositorios
- Permita realizar contribuciones en los repositorios
- Permita definir los procesos de contribución en los repositorios
- Permita compartir los derechos de confirmación o aceptación de cambios

Los sistemas de control de versiones manejan términos en común entre los cuales tenemos los siguientes (Davis & Daniels, 2016, pág. 179):

Commit

Un commit es un conjunto de acciones que corresponden al número total de cambios realizados en los archivos que están bajo el control de versiones.

Conflictos

Un conflicto es generado cuando se realizan dos cambios similares al mismo tiempo en un archivo, y por lo tanto, el sistema de control de versiones no puede determinar el cambio correcto para su aceptación y en la mayoría de las herramientas de control de versiones existe una forma de observar y seleccionar el cambio que se desea realizar o confirmar.

Petición de extracción (Pull request)

Una solicitud de extracción es un proceso que permite a los desarrolladores o usuarios del sistema de control de versiones indicar que un cambio está listo para ser revisado y fusionado con la rama principal.

Existen muchas herramientas en el mercado que cumplen con las características antes mencionadas, y en el presente trabajo se recurre al uso de la herramienta Github que se basa en Git y es una de las más utilizadas en la industria. Como característica importante de Github es la facilidad que tiene para integrarse a muchas herramientas como por ejemplo Jenkins, y tiene una documentación muy clara que ayudará a resolver cualquier problema que se presente a lo largo del desarrollo del trabajo al momento de la construcción del pipeline CI/CD.

Git

Es un software cuya función es mantener un control de versiones del código permitiendo encontrar todos los cambios que se realizan en un archivo o carpeta.

Permite llevar un seguimiento cuando se construye una aplicación y es de gran ayuda para el trabajo colaborativo ya que permite compartir o añadir cambios sin afectar el código principal.

Github

Es una plataforma que gestiona todo el código para la construcción de aplicaciones que utilizan el software Git y permite alojar código en la nube.

Ofrece una serie de herramientas muchas de las cuales son de pago, pero en su versión gratuita ofrece toda la funcionabilidad necesaria para llevar un correcto control de versiones de un proyecto.

Algunas de sus ventajas según (Soni, 2016, pág. 34) son:

- Los desarrolladores tienen la posibilidad de compartir su código y trabajar en forma simultánea.
- El código puede recuperarse en caso de pérdida por algún evento emergente como por ejemplo un daño en las máquinas de los desarrolladores. Ya que todo el código se guarda en servidores alojados en la nube.
- La curva de aprendizaje es pequeña y su rendimiento es bueno.
- Dispone de una serie de herramientas para facilitar el control de las versiones de un proyecto.

3.5.5. Herramientas para implementar un ciclo de integración y entrega continua

Composer

Es un gestor de dependencias desarrollado para aplicaciones que utilizan el lenguaje programación PHP (Análogo a npm para JavaScript o pip para Python).

Permite llevar un control sobre librerías que son desarrolladas por programadores externos y que se incorporan a un proyecto.

Jenkins

Es un servidor de automatización de código abierto y utilizado para implementar ciclos CI/CD.

Entre sus características importantes están las siguientes:

- Es multiplataforma y puede utilizarse en sistemas como Windows, Ubuntu/Debian, RedHat/Fedora, Mac OS X, openSUSE y FreeBSD.
- Se integra y puede funcionar con nubes públicas tales como Aws, Azure, etc. Además tiene muchos tipos de plugins que ayudan a mejorar el flujo de integración y entrega continua.

Ansible

(RedHat, 2020) Nos describe que "Ansible es un motor open source que automatiza los procesos para preparar la infraestructura, gestionar la configuración, implementar las aplicaciones y organizar los sistemas, entre otros procedimientos de TI".

Como principal característica que tiene Ansible es la de no utilizar un Daemon, ni base de datos adicional. Por lo tanto, se puede trabajar desde cualquier editor de texto y llevar un control de versiones con Git.

Automatización de la infraestructura y Terraform

La automatización de la infraestructura es el proceso que consiste en aprovisionar elementos de infraestructura a través de código, con la posibilidad de recuperar el negocio en el caso de alguna falla realizando copias de seguridad de los datos, el repositorio donde se aloja el código y los recursos informáticos (Davis & Daniels, 2016, pág. 182).

Al definir toda la infraestructura mediante código se facilita llevar un control de los cambios realizados, además permite replicar la infraestructura de forma rápida, y de ser necesario tener plantillas que se puedan reutilizar para implementar otros proyectos.

Terraform es una herramienta de código abierto de Infraestructura como código, creada por HashiCorp. Utiliza una codificación declarativa que permite a los desarrolladores utilizar un lenguaje de configuración de alto nivel llamado HCL (HashiCorp Configuration Language) para describir la infraestructura en un entorno de cloud y utiliza una sintaxis simple y puede suministrar la infraestructura en centros de datos tanto en un entorno local como cloud.

Vagrant

Tener un ecosistema local para realizar pruebas es muy importante como lo menciona (Davis & Daniels, 2016, pág. 177), un entorno de desarrollo local sólido es esencial para que las personas encargadas de implementar un proyecto empiecen a contribuir al producto. Por lo tanto tener las herramientas necesarias para la creación del entorno hará que la implementación de nuevas características se cumpla de manera efectiva.

Los requisitos mínimos necesarios pueden variar de acuerdo a los gustos personales, y definir las herramientas comunes para la implementación de un entorno de desarrollo local incluye determinar si existe un marco coherente que los equipos comparten. "La coherencia

de las experiencias puede agilizar y facilitar la incorporación de nuevos empleados y que estos empiecen a hacer contribuciones significativas" (Davis & Daniels, 2016, pág. 178).

Hay que tener un equilibrio que permita garantizar una experiencia coherente al personalizar un puesto de trabajo en conjunto con las herramientas utilizadas, priorizando las herramientas que se adapten mejor a las necesidades individuales. Pero hay que tener cuidado ya que un exceso de personalización puede producir un bloqueo debido a la sobrecarga adicional de tiempo y esfuerzo que son utilizados para la configuración de entornos especializados. (Davis & Daniels, 2016, pág. 179)

Para implementar un entorno de desarrollo local se utiliza Vagrant que es una herramienta de código abierto que brinda entornos de desarrollo portables. Puede trabajar con otros programas tales como: VMware, VirtualBox, Hyper-V, KVM, AWS y contenedores Docker.

Vagrant está desarrollado en lenguaje Ruby, y puede utilizar otros lenguajes de programación. Los archivos de configuración de Vagrant son llamados Vagrantfiles y pueden ser compartidos para replicar máquinas virtuales ya creadas.

Proveedores de nube

En el mercado existen muchos proveedores de nube, pero para el presente proyecto se utiliza Amazon Web Services (AWS).

AWS es una plataforma muy popular en el mundo, que ofrece más de 200 servicios integrales de centros de datos a nivel global. Cuenta con Millones de clientes y existe extensa documentación que ayudan a resolver los problemas que se puedan presentar.

Algunos de los servicios que ofrece la plataforma AWS se visualizan en la Figura 7, para el presente trabajo se utilizará el servicio de instancias EC2, que son máquinas virtuales que se pueden crear y desplegar, además, cuentan con opciones de configuración que ayudan a mejorar la seguridad en la infraestructura, como es el caso de los grupos de seguridad, que permiten o restringen el paso de tráfico dependiendo de las políticas configuradas.

Figura 7. *Nombre de los servicios que ofrece AWS*

	AWS
Virtual machines	Amazon EC2
PaaS	Elastic Beanstalk
Container services	Amazon EC2 Container Services
RDBMS	Amazon RDS
NoSQL	DynamoDB
BIG Data	Amazon EMR
Networking	Amazon VPC
Cache	Amazon Elasticache
Import/export	Amazon import/export
Search	Amazon CloudSearch
CDN	CloudFront
Identity and access management	AWS IAM and Directory Services
Automation	AWS OpsWorks

(AWS, 2022)

También se hará uso del servicio de Amazon VPC que nos permitirá definir una red privada virtual que alojara a la aplicación web, y dentro de dicha VPC se consumirá los recursos o servicios de AWS nos brinda tales como grupos de seguridad, instancias EC2, etc.

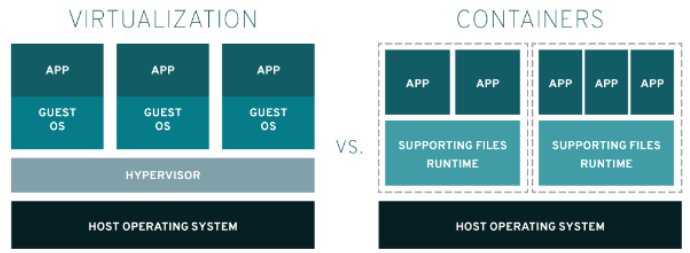
3.5.6. Tecnología de contenedores Docker

Docker es una herramienta de código abierto creada para envolver el código, el entorno de ejecución, las herramientas del sistema y bibliotecas. Los contenedores Docker comparten el kernel en donde se ejecutan y, como resultado se inician al instante. Los contenedores Docker se ejecutan tanto en Windows como en distribuciones Linux, pero en el caso de Windows hay que realizar una serie de pasos y configuraciones adicionales para que pueda funcionar. (Soni, 2016, pág. 48)

Para comprender la diferencia entre una máquina virtual y un contenedor se puede explicar mediante la Figura 8. Pero como diferencia principal se menciona que las máquinas virtuales necesitan de un Hipervisor para administrar y dividir todos los recursos que tiene una máquina, por lo tanto cada máquina virtual tiene su propio sistema operativo y reserva una porción de hardware. A diferencia de los contendores, que comparten un mismo Kernel o

núcleo del sistema operativo así como también los recursos entre todos los contenedores que se ejecutan en la maquina Host.

Figura 8. *Diferencias entre contenedores y virtualización*



Virtualización

El software llamado hipervisor separa los recursos de las máquinas físicas para que puedan dividirse y destinarse a las máquinas virtuales. Cuando un usuario emite una instrucción de una máquina virtual que requiere recursos adicionales del entorno físico, el hipervisor transmite la solicitud al sistema físico y almacena los cambios en la memoria caché. Las máquinas virtuales parecen servidores físicos y actúan como tales, lo que puede multiplicar las desventajas de las dependencias de las aplicaciones y los grandes footprints del sistema operativo. Incluso, en la mayoría de los casos, ese footprint no se necesita para ejecutar una sola aplicación o microservicio.

Contenedores

Los contenedores contienen un microservicio o una aplicación y todo lo que necesita para ejecutarse. Lo que está dentro de un contenedor se conserva en lo que llamamos imagen: un archivo basado en un código que incluye todas las bibliotecas y las dependencias. Estos archivos pueden considerarse una instalación de la distribución de Linux, porque la imagen viene con paquetes RPM y archivos de configuración. Como los contenedores son tan pequeños, suele haber cientos de ellos con un nivel de acoplamiento bajo, así que se utilizan plataformas de organización de contenedores (como Red Hat OpenShift y Kubernetes) para configurarlos y gestionarlos.

(RedHat, 2020)

Docker Compose

Es una herramienta diseñada para definir y ejecutar aplicaciones Docker de varios contenedores. Utiliza un archivo YAML para configurar los servicios que se desplegaran con Docker. Y mediante un comando, crea e inicia todos los servicios desde cero.

4. Descripción del sistema desarrollado / Implementación

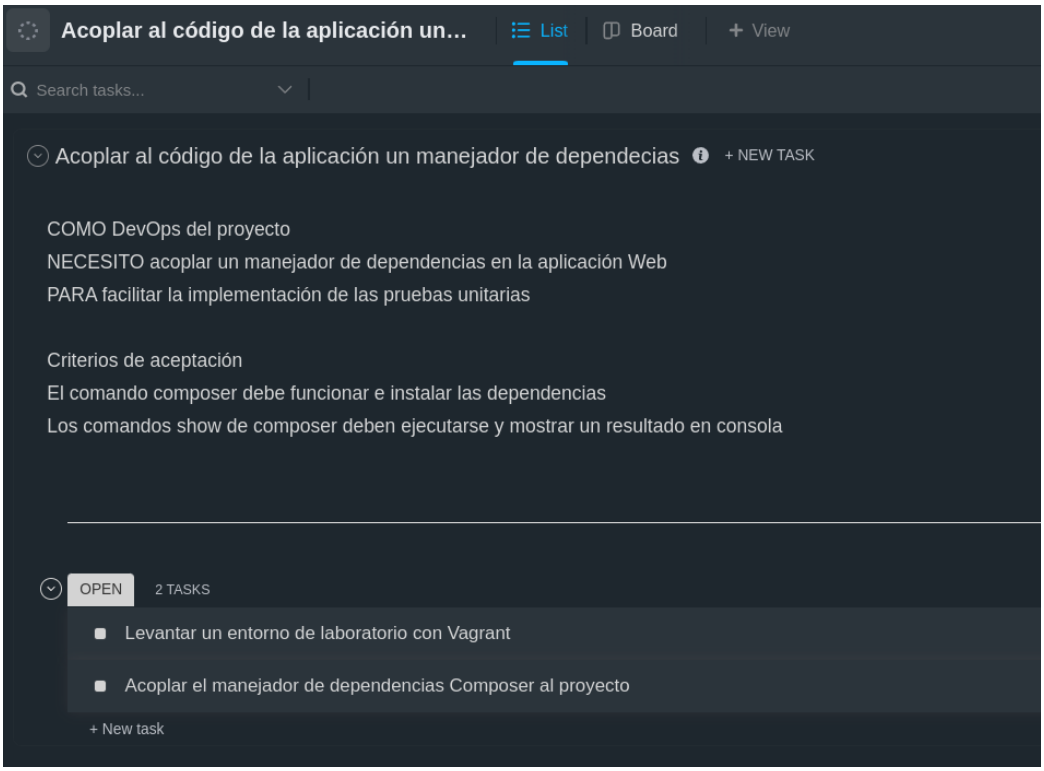
Listado de objetivos y requisitos

De acuerdo a los requisitos y para alcanzar los objetivos planteados se crean 6 historias de usuario que a su vez se dividen en tareas como se muestra a continuación utilizando la herramienta de control de proyectos ClickUp.

Definir e ingresar las historias de usuario a la plataforma ClickUp

4.1. Acoplar al código de la aplicación un manejador de dependencias

Figura 9. *Acoplar al código de la aplicación un manejador de dependencias*



Fuente (Autoría propia)

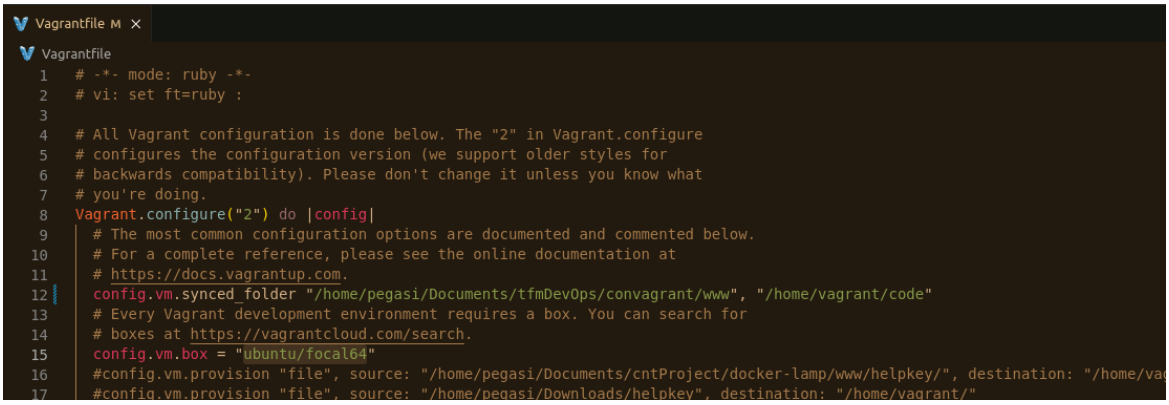
La primera tarea indispensable para implementar un ciclo CI/CD en el proyecto corresponde a la construcción de pruebas unitarias en el código de la aplicación web como se muestra en la Figura 9, y para cumplir con este objetivo es necesario tener un control y orden de las librerías que conforman la aplicación. Por lo tanto se integra un manejador de dependencias al código de la aplicación.

La aplicación está construida con el lenguaje de programación PHP y no utiliza ningún framework. De este modo es necesario acoplar de forma manual Composer al proyecto, para facilitar el manejo e instalación de las librerías necesarias en el proyecto.

4.1.1. Levantar un entorno de laboratorio con Vagrant

Para integrar el manejador de paquetes Composer al proyecto se procede a levantar un entorno de laboratorio por medio de la herramienta Vagrant bajo un sistema operativo Ubuntu 20.04 LTS como se muestra en la Figura 10.

Figura 10. *Archivo de configuración de Vagrant*



Fuente (Autoría propia)

Se utiliza la opción de sincronización de carpetas entre la máquina local y la máquina creada mediante la herramienta Vagrant, para no gastar tiempo innecesario al realizar una conexión SSH, y poder manipular los archivos que conforman la aplicación cada vez que se necesite realizar cambios en el código.

En la máquina con sistema operativo Ubuntu 20.04 LTS es necesario instalar paquetes previos que permitirán la correcta instalación y funcionamiento del manejador de dependencias para Php, y son los siguientes:

- **Curl**

  Permite la trasferencia o descarga de archivos

- **Php Cli**

  Permite ejecutar comandos en php

- **Php-mbstring**

  Proporciona funciones necesarias para la instalación de Composer

- **Git**

  Programa de control de versiones

Para realizar la instalación de los paquetes ejecutamos el siguiente comando en la consola de Ubuntu:

sudo apt install curl php-cli php-mbstring git

4.1.2. Acoplar el manejador de dependencias Composer al proyecto

Una vez instalados todos los paquetes necesarios se procede a realizar la instalación de la herramienta Composer de acuerdo a los pasos que se detallan en su página oficial

https://getcomposer.org/download/

Nos colocamos en la carpeta raíz del código de la aplicación y se procede a iniciar Composer en el proyecto mediante comandos de consola.

Para cumplir con los dos criterios de aceptación se procede a validar la creación del archivo composer.json en el cual se especifica todas las librerías que se instalaran en el proyecto.

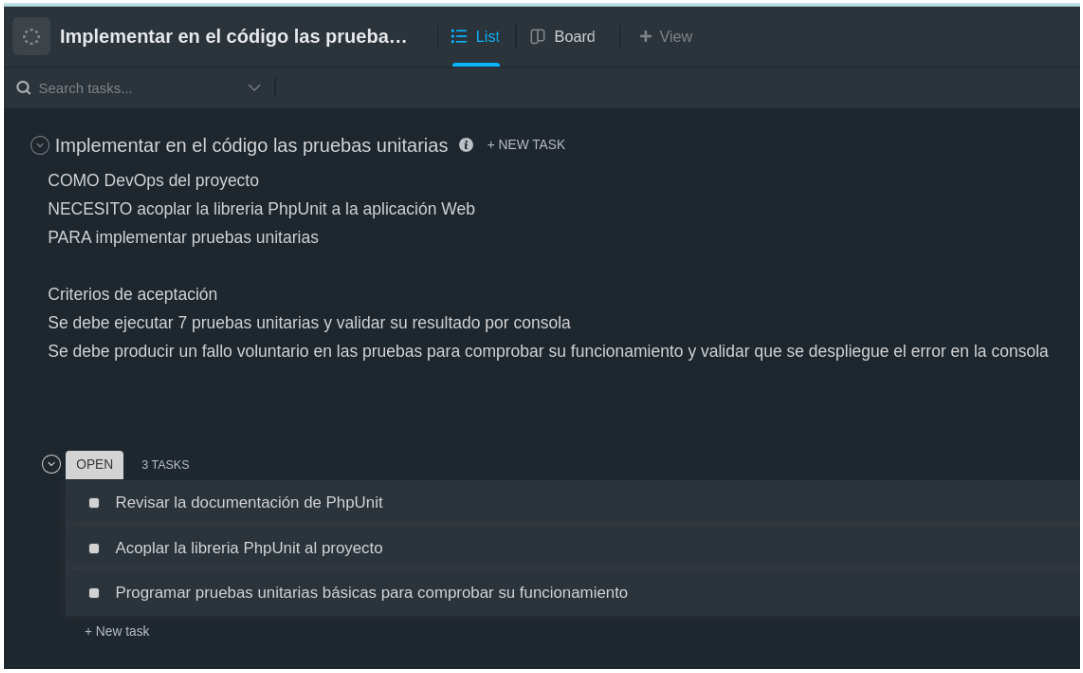
Se aplica comandos show de Composer para validar su versión y los paquetes disponibles como se muestra en la Figura 11.

Figura 11. *Comandos show de Composer*

Fuente (Autoría propia)

4.2. Implementar en el código las pruebas unitarias

Figura 12. *Implementar en el código las pruebas unitarias*



Fuente (Autoría propia)

Una vez se valida el correcto funcionamiento de Composer es necesario instalar una librería para facilitar la construcción de pruebas unitarias en el proyecto, ya que al momento no están implementadas, y esta historia es ingresada a la plataforma de seguimiento como se muestra en la Figura 12.

4.2.1. Revisar la documentación de PhpUnit

De acuerdo a la documentación para realizar las pruebas unitarias se debe tener dos carpetas en el proyecto de la siguiente manera:

Carpeta src

En esta carpeta se implementa todas las funciones que serán testeadas por pruebas unitarias

Carpeta test

La carpeta contiene las pruebas unitarias que se realizan a las funciones implementadas en la carpeta src. Como recomendación de acuerdo a la documentación se menciona que el nombre del archivo que contiene las pruebas unitarias debe tener el mismo nombre del

archivo que contiene las funciones pero aumentando la palabra test como se muestra a continuación:

Carpeta Src

operacionesHelpact.php

Carpeta test

operacionesHelpactTest.php

4.2.2. Acoplar la librería PhpUnit al proyecto

Para acoplar la librería al proyecto se ejecuta el comando composer require phpunit/phpunit y como resultado se agrega la librería.

4.2.3. Programar pruebas unitarias básicas para comprobar su funcionamiento

Finalmente probamos los test unitarios mediante el comando ./vendor/bin/phpunit ubicándonos en la raíz del proyecto como se muestra en la Figura 13.

Figura 13. *Programar pruebas unitarias básicas*

Fuente (Autoría propia)

En la Figura 13 se visualiza que se cumplió el primer criterio de aceptación y se realizaron 7 test y todos son correctos.

A continuación, se genera 2 fallos voluntarios y se visualiza el error por consola como se muestra en la Figura 14.

Figura 14. *Errores en las pruebas unitarias*

Fuente (Autoría propia)

4.3. Escoger las herramientas DevOps para realizar el pipeline CI/CD

Figura 15. *Herramientas DevOps para realizar el pipeline CI/CD*

Fuente (Autoría propia)

Para cumplir con el objetivo de migrar la aplicación a un proveedor de nube e implementar un ciclo CI/CD es necesario tener un conjunto de herramientas que funcionen entre si y poder cumplir con cada uno de las etapas necesarias para entregar un producto final. La historia es ingresada a la plataforma de seguimiento como se muestra en la Figura 12.

4.3.1. Investigar sobre las herramientas más utilizadas en la creación de pipelines CI/CD

Según el objetivo planteado y para cumplir con la solución al problema planteado se escoge las siguientes herramientas de acuerdo a la Tabla 3.

Tabla 3. *Herramientas DevOps*

Herramientas DevOps para automatizar un ciclo de vida de la aplicación	
Función	Nombre de la herramienta
Infraestructura como Código	Terraform
Integración despliegue Continuo	Jenkins
Tecnología de contenedor	Docker
Proveedor de nube	AWS
Administración de la configuración	Ansible

Fuente (Autoría propia)

El criterio para escoger las herramientas se realiza en base a la disponibilidad de la documentación e información en el internet, ya que al ser altamente utilizadas se puede encontrar con facilidad información que ayuda a resolver los retos que se presenten al momento de implementar la pipeline CI/CD.

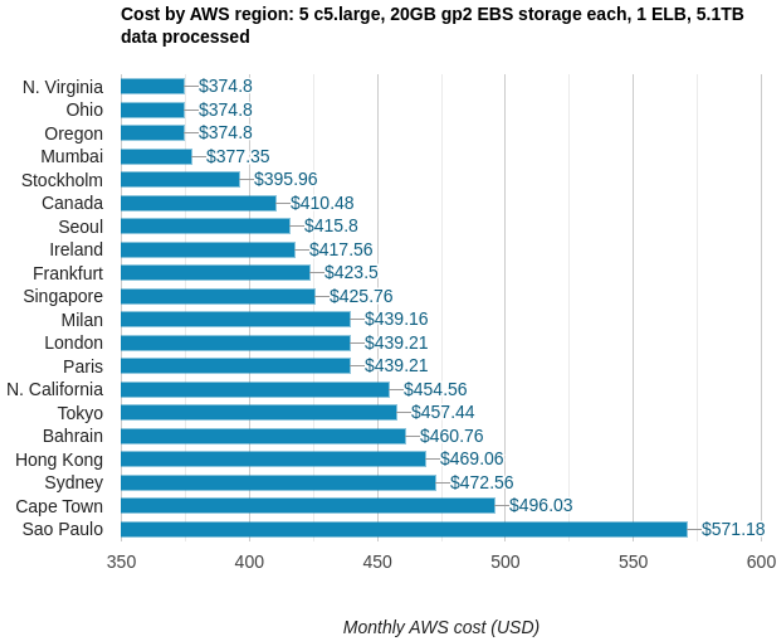
4.3.2. Analizar y escoger la zona de disponibilidad del proveedor AWS

El costo de la implementación de los servicios que ofrece AWS varía de acuerdo a la zona de disponibilidad. Por lo tanto esto se debe tomar en cuenta al momento del despliegue de la infraestructura que sostiene el presente proyecto.

Según (Márquez, 2022) “Elegir una región de AWS es la primera decisión que debe tomar al configurar componentes de AWS. La mayoría de los clientes de AWS eligen uno en función de la proximidad con ellos mismos o con sus usuarios finales”

Y en base a la Figura 16:

Figura 16. *Precio del envío de tráfico ELB a 5 instancias c5.large que ejecutan Amazon Linux en la misma zona de disponibilidad*



(Márquez, 2022)

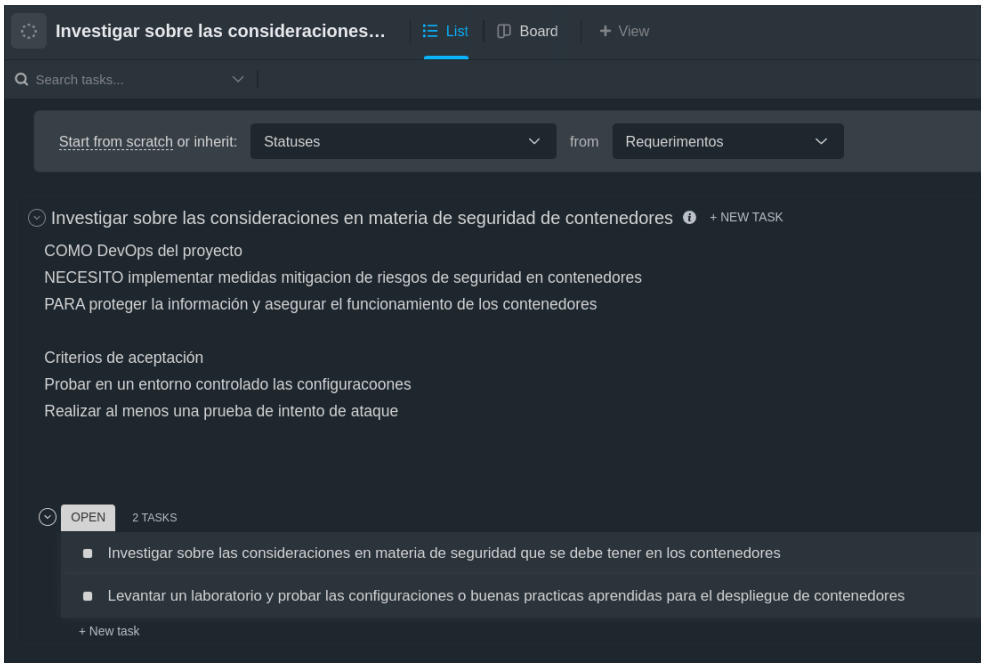
Se visualiza que las regiones en donde los servicios o la trasferencia de datos son más baratos son las regiones de N Virginia, Ohio y Oregon.

Por motivos de proximidad se avaluó la posibilidad de utilizar la zona de Sao Paulo pero el costo en mucho mayor por lo que se toma la decisión de utilizar la región de Ohio.

Mediante el uso de todas las herramientas que tienen muchos años en el mercado se cumple con el criterio de aceptación y se valida en su documentación que pueden trabajar en conjunto para construir un pipeline CI/CD.

4.4. Investigar sobre las consideraciones en materia de seguridad de contenedores

Figura 17. *Consideraciones en materia de seguridad de contenedores*



Fuente (Autoría propia)

4.4.1. Investigar sobre las consideraciones en materia de seguridad que se debe tener en los contenedores

Cambio de usuario root al crear una imagen

En base a la investigación realizada a partir del ataque producido a un contenedor de pruebas que estaba alojado en el proveedor de nube Digital Ocean que como resultado desencadeno en el bloqueo y desconexión de la red de acuerdo a la Figura 1 y cumplir con los requerimientos de la historia de usuario de la Figura 17.

Se identifica que al construir las imágenes Docker y utilizar los ficheros Dockerfile por defecto, como resultado se generan imágenes Docker utilizando el usuario root. Entonces, se produce una vulnerabilidad ya que al ejecutarse como usuario root se tiene acceso a todo el contenedor con la posibilidad de modificar sus recursos. Así como también, el atacante puede inyectar código malicioso y ejecutar comandos shell o montar volúmenes con los permisos de root. (Encora, 2020)

Para evitar este escenario se deben crear nuestras propias imágenes Docker y configurar un usuario diferente al root

Validar los accesos y las contraseñas por defecto que tienen las imágenes de Docker hub

Otro escenario identificado a tener a consideración es que al utilizar una imagen pre fabricada de Docker hub es necesario leer la documentación a profundidad para validar todas las claves por defecto que tienen, ya que si no las cambiamos o si no sabemos de su existencia puede producirse una vulnerabilidad, para el caso del contenedor vulnerado se identificó que el acceso por llaves ssh estaba activo pero por medio de un usuario y contraseña por defecto por lo cual fue vulnerado.

Vulnerabilidad al conceder permisos de root por defecto al comando Docker

Según la documentación oficial de Docker (Docker, 2021) Ejecutar contenedores implica ejecutar el daemon de Docker y requiere privilegios root que solo los usuarios de confianza deben poder controlar. También menciona que "Docker permite compartir un directorio entre el host de Docker y un contenedor invitado; y permite hacerlo sin limitar los derechos de acceso del contenedor".

Por lo tanto no es recomendable dar los permisos de root al comando Docker por lo tanto siempre se debe anteceder la palabra sudo para ejecutar los comandos.

En el caso de tener un entorno de pruebas o para aprendizaje se puede utilizar que para entornos de producción no es recomendable.

La documentación oficial de Docker también menciona que "los contenedores Docker son, por defecto, bastante seguros; especialmente si ejecuta sus procesos como usuarios sin privilegios dentro del contenedor".

Procedencia de las imágenes de Docker

En el repositorio oficial de Docker existe gran cantidad de imágenes de dominio público, que facilitan el proceso de construcción de los contenedores o del ambiente en el cuál se desplegará una determinada aplicación, pero esto conlleva a serios problemas de seguridad ya que las imágenes pueden contener algún tipo de vulnerabilidad o virus malicioso.

Según (Google Cloud, 2018) algunas de las razones que nos impiden la utilización de imágenes públicas son las siguientes:

- Existe la necesidad de controlar y saber el contenido exacto dentro de la imagen
- Se desea eliminar la dependencia hacia un repositorio externo
- Existe la necesidad de controlar o mitigar vulnerabilidades en el entorno de producción

Por lo tanto es recomendable construir las imágenes desde cero en el caso de plataformas críticas o utilizar un sistema mixto mediante el uso de imágenes base de repositorios oficiales y sobre estas construir todo lo necesario.

Procedencia del Dockerfile

Las siguientes recomendaciones según (Rice, 2020, pág. 75) ayudan a mejorar la seguridad en la construcción de imágenes generadas por medio de un archivo Dockerfile, y reduce la posibilidad de ataques que puedan comprometer los contenedores que utilizan dichas imágenes.

Imágenes base

- Se debe utilizar una imagen base de un registro de confianza, ya que imágenes creadas por terceras personas u organizaciones pueden incluir código malicioso.
- La imagen base debe ser lo más optimizada posible evitando que la superficie de ataque sea menor, por lo tanto es altamente recomendable evaluar la posibilidad de construir la imagen desde cero o utilizar una imagen base como distroless (https://github.com/GoogleContainerTools/distroless) que se caracteriza por no contener administradores de paquetes, shells ni ningún otro programa que viene instalado por defecto en una distribución estándar de Linux. De este modo se tiene la ventaja de generar imágenes de tamaño reducido que contienen solo lo necesario y su despliegue es rápido.

La instrucción User no deber ser Root

- La instrucción USER utilizada por Dockerfile define el usuario por defecto para ejecutar los contenedores, por lo tanto es recomendable especificar un usuario no root en todos los Dockerfiles.

Comandos Run

- Un comando RUN especificado dentro de un archivo Dockerfile puede ejecutar cualquier comando. Por tal razón, en el caso que un intruso comprometa el archivo Dockerfile, el atacante puede introducir y ejecutar cualquier código.
- Es necesario que los privilegios para la modificación de los archivos de Docker estén limitados a miembros de confianza, y hay que tomar mucha atención al código que se cambia. De tal modo es necesario solicitar una comprobación o auditoría cuando se modifiquen o introduzcan comandos RUN.

Volúmenes

- Por lo general al momento de realizar pruebas o construir demos se monta los directorios de la máquina host en el contenedor por medio de volúmenes. Entonces, es importante revisar que el archivo Dockerfile no monte directorios sensibles como /etc o /bin en el contenedor.

Datos sensibles en el archivo Dockerfile y código innecesario

- No se debe incluir contraseñas, credenciales o cualquier dato que sea secreto, ya que al momento de la construcción de la imagen estos quedan expuestos y se genera una potencial vulnerabilidad.
- La imagen debe contener solo el código necesario para el funcionamiento de la aplicación o servicio, lo que se traduce en menos código dentro del contenedor y menos código vulnerable.

Incluir todo lo necesario en el contenedor

- Incluir todas las dependencias necesarias en la imagen evita la instalación de paquetes adicionales en tiempo de ejecución del contenedor, ya que si realizamos la instalación de paquetes que no están incluidos en el dockerfile, aumenta la posibilidad de instalar un paquete comprometido en su seguridad.

Diferencia entre seguridad en las aplicaciones y seguridad en los contenedores

(Wetter, owasp, 2022) Informa que Docker, como cualquier tecnología de contenedores, no resuelve los problemas de seguridad en las aplicaciones. “No ayuda a realizar una validación de entrada y no proporciona protección contra la inyección SQL”.

La seguridad de los contenedores está enfocada con la seguridad del sistema, la red y con el diseño arquitectónico.

Escaneo de vulnerabilidades para imágenes de Docker

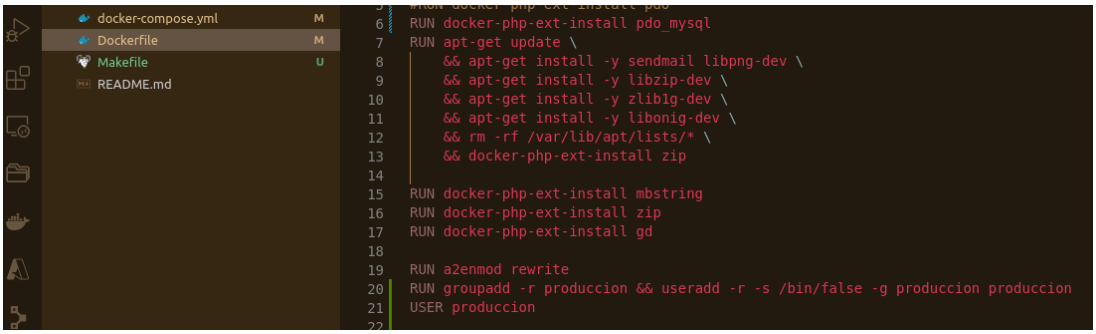
Existen varias herramientas en el mercado que realizan una revisión de las imágenes de Docker, las comparan en sus bases de datos de vulnerabilidades y nos brinda los resultados de todas las fallas de seguridad encontradas al interior de la imagen. Son herramientas de gran ayuda y algunas están disponibles como proyectos Open Source pero se recomienda contratar un servicio de escaneo de vulnerabilidades.

Docker cuenta con su propia herramienta para el análisis de vulnerabilidades que permite a los desarrolladores y sus equipos revisar la seguridad de las imágenes para implementar medidas que solucionen los problemas identificados en el análisis. De este modo, se generan imágenes de Docker más seguras (Docker, 2021).

4.4.2. Levantar un laboratorio y probar las configuraciones o buenas prácticas aprendidas para el despliegue de contenedores

Se levanta un laboratorio local y se prueba las configuraciones, el despliegue y el correcto funcionamiento de los contenedores como se muestra en la figura.

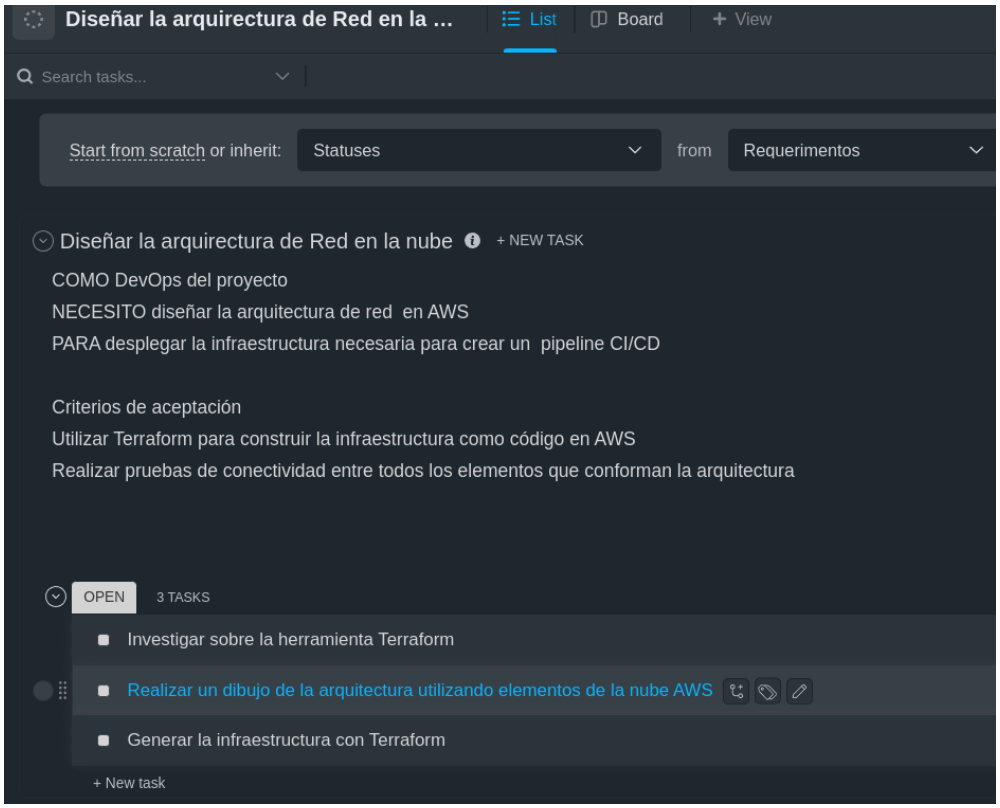
Figura18. *Configuraciones o buenas prácticas aprendidas para el despliegue de contenedores*



Fuente (Autoría propia)

4.5. Diseñar la arquitectura de Red en la nube

Figura 19. *Arquitectura de Red en la nube*



Fuente (Autoría propia)

4.5.1. Investigar sobre la herramienta Terraform

Terraform es una herramienta de infraestructura como código que permite la construcción, el cambio y llevar un control de versiones de una infraestructura.

Se toma como referencia la documentación oficial de la herramienta que se encuentra en la dirección https://www.terraform.io/docs para poder encontrar todas las funcionalidades que brinda y poder realizar un código más limpio utilizando funciones propias de la solución.

4.5.2. Realizar un dibujo de la arquitectura utilizando elementos de la nube AWS

Se define una arquitectura en Aws por medio de la creación de una Red privada virtual (VPC) que tiene dos grupos de seguridad. El grupo de seguridad 1 (sg1) tiene salida a internet y es la que engloba a las instancias EC2 que permiten el despliegue de la pipeline CI/CD, la cual está compuesta de acuerdo al siguiente cuadro:

Tabla 4. *Distribución de las Instancias*

Distribución de las Instancias	
Grupo de seguridad 1	**Grupo de seguridad 2**
Servidor de Jenkins	Máquina Bastión host
Servidor Controlador Ansible	
Servidor Nodo controlado Ansible	

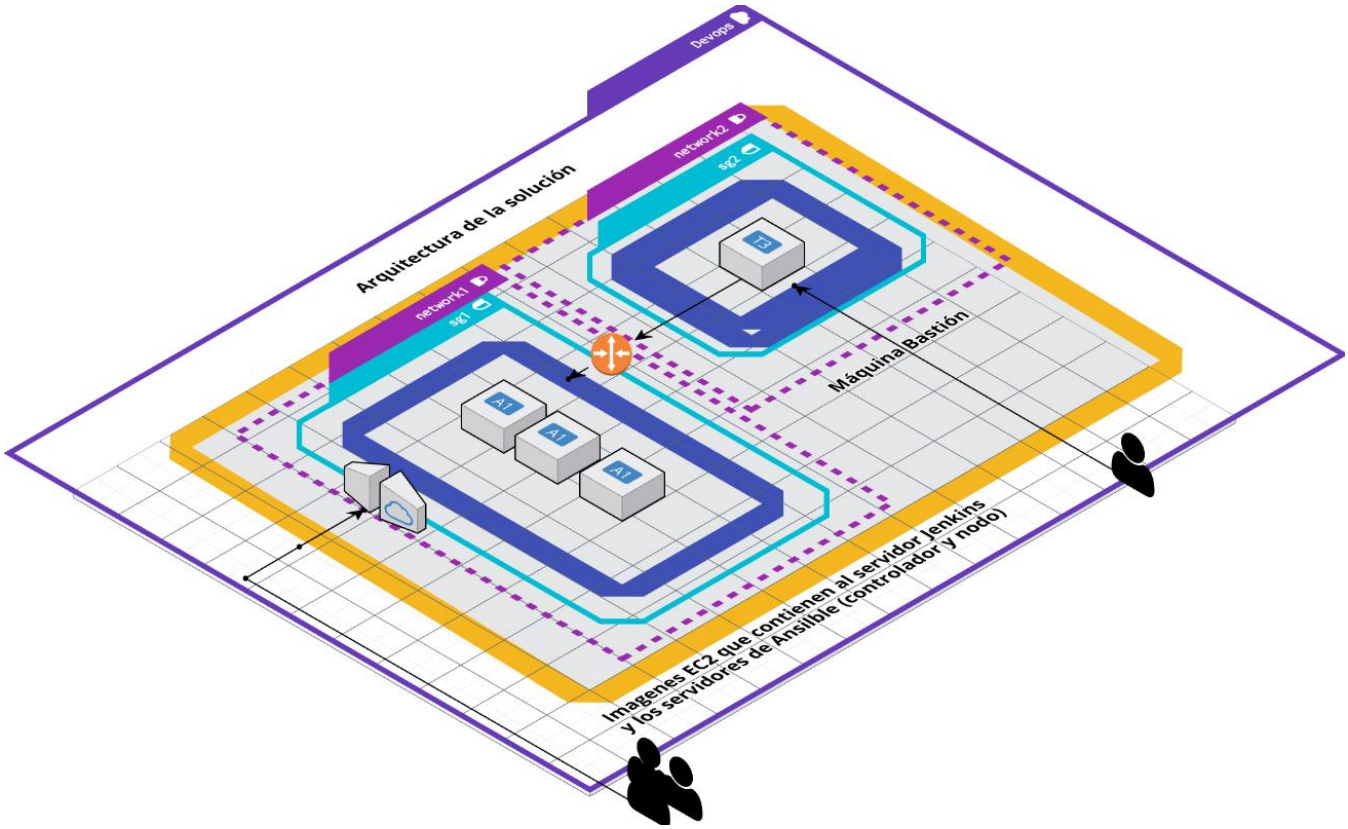
Fuente (Autoría propia)

El segundo grupo de seguridad máquina EC2 contiene una máquina bastion host (servidor bastión) que se define como un sistema informático que esta configurado de tal forma que resiste ataques y, de esta manera, proteger la red en la cuál esta localizado.

Para mayor seguridad se definen dos grupos de seguridad (sg1 y sg2), uno para todas las instancias que conforman el pipeline CI/CD y otro solo para la máquina bastión, y a su vez los grupos de seguridad tiene definidos cada uno una red que estan interconectadas entre si como se muetra en la Tabla 4.

El acceso ssh a las máquinas del grupo de seguridad 1 desde el exterior se encuentra restringido y se habilita la conexión ssh solamente desde al host bastión hacia toda la red. Por lo tanto, para ingresar a cualquien máquina se debe realizar un salto desde el bastión que previamente esta cargado con las llaves ssh instaladas en todas las instancias EC2 y que permiten su configuracion y acceso a permisos root.

Figura 20. *Arquitectura de la solución propuesta*



Fuente (Autoría propia)

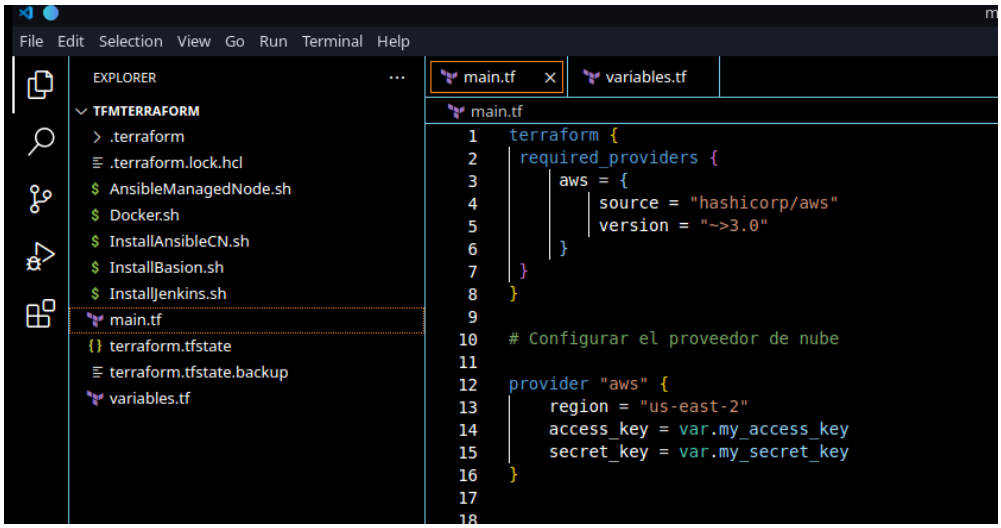
4.5.3. Generar la infraestructura con Terraform

Para el despliegue automático de la arquitectura es necesario utilizar una herramienta de infraestructura como código como Terraform ya que podemos trasladar la arquitectura propuesta a código y a su vez a la nube AWS.

Como medida de seguridad y según lo investigado es recomendable colocar las contraseñas de acceso a la nube en un archivo aparte y solo llamarlo cuando se produzca el despliegue.

Para generar la infraestructura se genera dos archivos (main.tf y Variables.tf), el archivo main.tf contiene todo el código que aprovisiona la infraestructura en la nube de AWS y por motivos de seguridad en el archivos variables.tf se encuentran las claves de acceso a la nube AWS. Como se muestra en la Figura 21.

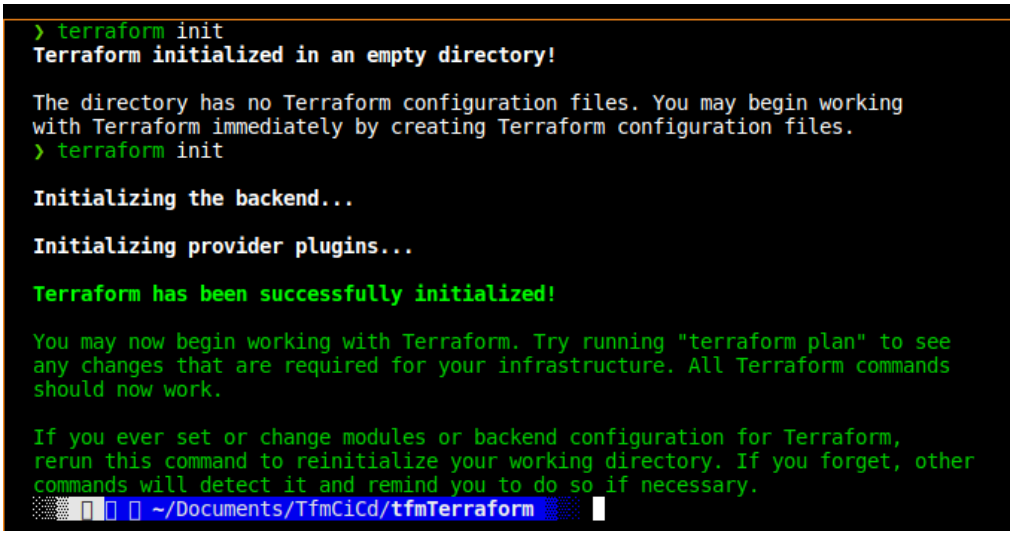
Figura 21. *Infraestructura con Terraform*



Fuente (Autoría propia)

Se aplica el comando **terraform init** para inicializar como se muestra en la Figura 22.

Figura 22. *Iniciar Terraform*



Fuente (Autoría propia)

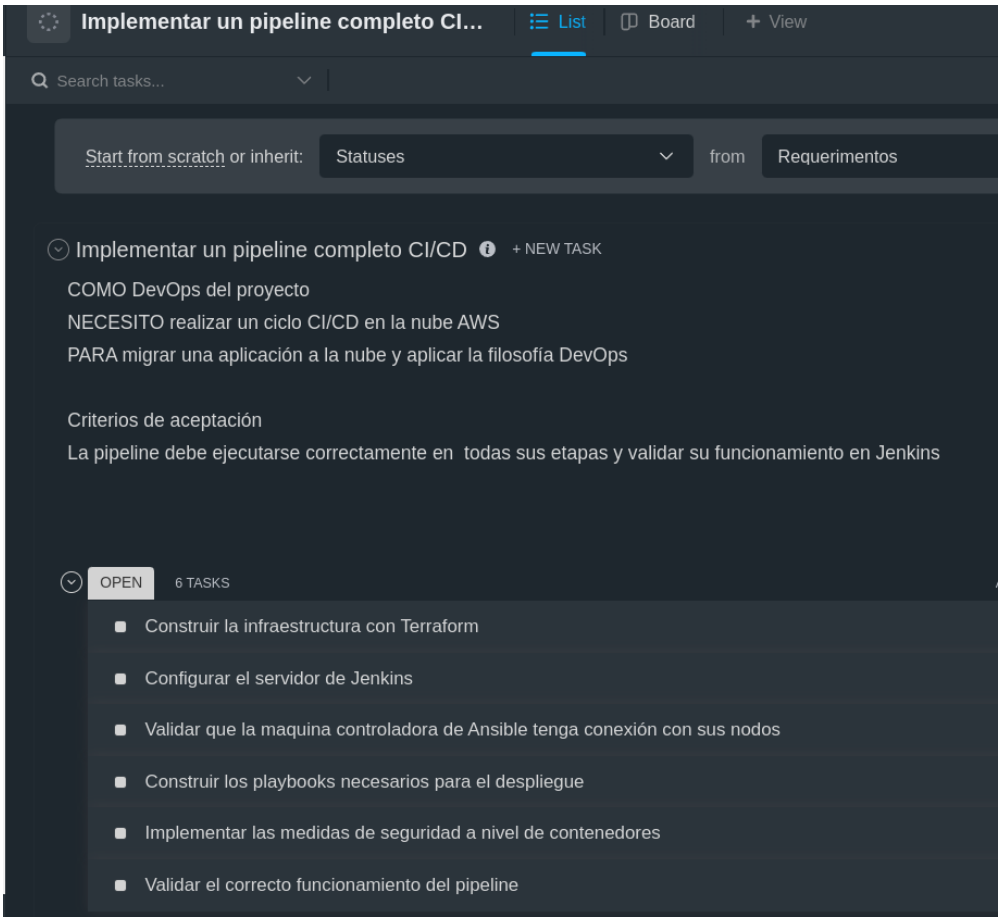
Antes del despliegue se utiliza el comando **terraform plan** que permite visualizar los cambios que se van a realizar en la nube, si por medio del análisis de la información desplegada, los cambios o configuraciones son las esperadas, se procederá a ejecutar el comando para la construcción de la infraestructura en la nube AWS.

4.5.4. Pruebas de conectividad entre todos los elementos que conforman la arquitectura

La arquitectura propuesta está conformada por dos grupos de seguridad (sg1 y sg2), en el sg1 se encuentran los elementos necesarios para que el pipeline funcione correctamente y en el sg2 se encuentra el bastión host que permitirá el acceso a todas las instancias EC2 de la arquitectura. Por motivos de prueba y configuración del pipeline se habilita el protocolo ICMP en los grupos de seguridad, para permitir que el comando ping envíe solicitudes eco ICMP a las direcciones ip privadas correspondientes a cada instancia, y de este modo verificar la conectividad entre todas los elementos.

4.6. Implementar un pipeline completo CI/CD

Figura 23. *Tarea para construir un pipeline completo CI/CD*

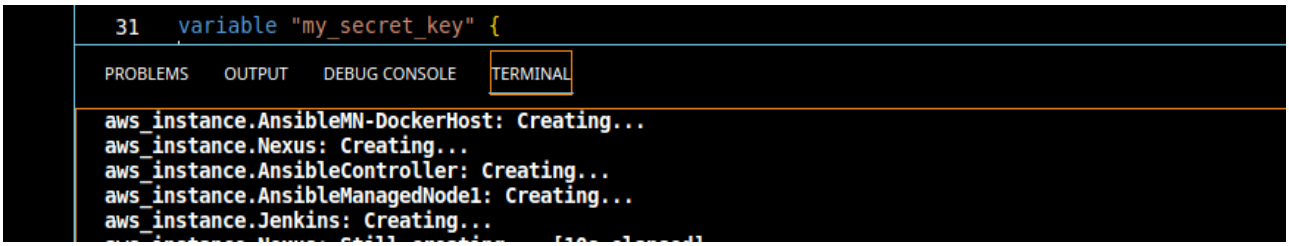


Fuente (Autoría propia)

4.6.1. Construir la infraestructura con Terraform

Se ingresa la historia de usuario a la plataforma de seguimiento como se muestra en la Figura 23, y cuando las configuraciones a realizar son analizadas y validadas se ejecuta el comando **terraform apply --auto-approve** que realiza los cambios sin necesidad de confirmarlo por consola como se muestra en la Figura 24.

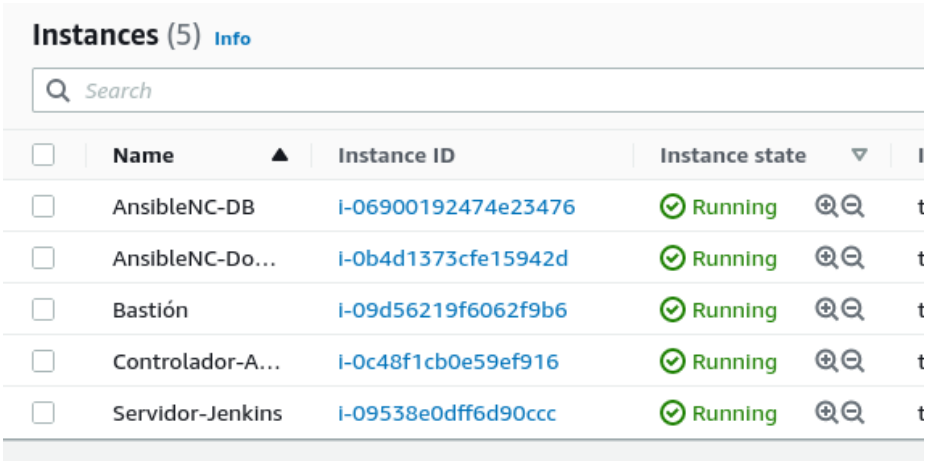
Figura 24. *Construcción la infraestructura con Terraform*



Fuente (Autoría propia)

Como resultado se crean las instancias en AWS de acuerdo a la arquitectura planteada como se muestra en la Figura 25.

Figura 25. *Instancias EC2 creadas en AWS por medio de Terraform*



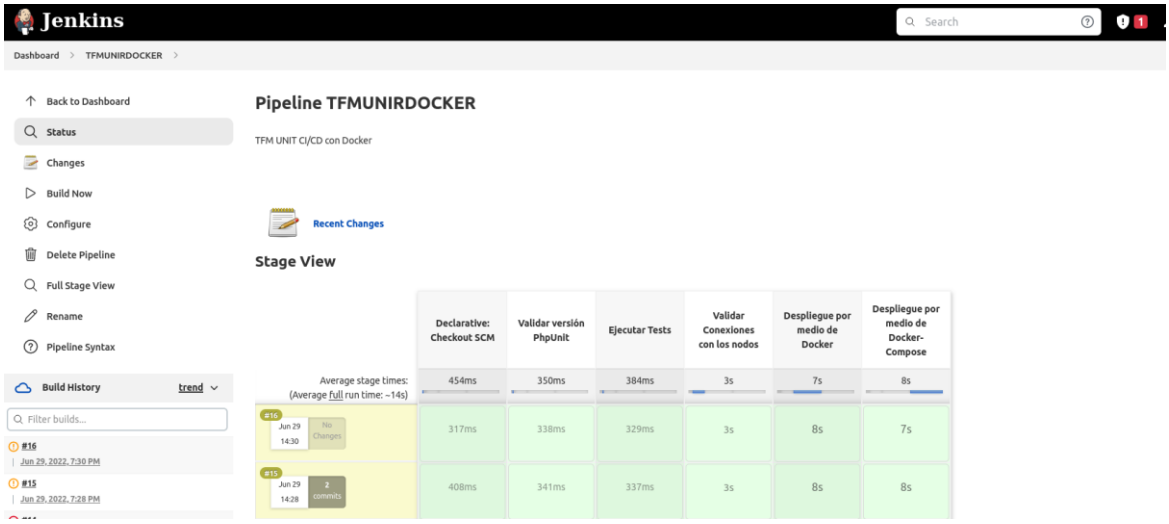
Fuente (Autoría propia)

4.6.2. Configurar el servidor de Jenkins

Por medio de Terraform se ejecuta un comando de instalación de Jenkins en el servidor y una vez instalado se ingresa a la plataforma y se configura un pipeline conformado por 5 etapas como se muestra en la Figura 26 y son:

- Validar versión PhpUnit
- Ejecutar Tests
- Validar Conexiones con los nodos
- Despliegue por medio de Docker
- Despliegue por medio de Docker-Compose

Figura 26. *Pipeline CI/CD creado en Jenkins*



Fuente (Autoría propia)

El pipeline se conectar a Github para acceder al repositorio de la aplicación y del archivo de docker compose como se muestra en la Figura 27.

Figura 27. *Conexión de Jenkins con el repositorio de la aplicación*

Fuente (Autoría propia)

4.6.3. Validar que la máquina controladora de Ansible tenga conexión con sus nodos

Se configura el servidor controlado de Ansible y se establece conexión con el nodo por medio de llaves ssh una vez se comprueba el acceso, se procede a configurar un playbook para validar la conectividad como se muestra en la Figura 28.

Figura 28. *Validar conexión de los nodos*

```
[ansibleadmin@ip-172-20-10-217 playbooks]$ ansible-playbook validarConexionServers.yaml  -i hosts

PLAY [Validar conexion de los servidores de Docker] ****************************************************

TASK [Gathering Facts] *********************************************************************************
ok: [localhost]

TASK [Validar conexion con los nodos] ******************************************************************
changed: [localhost]

PLAY RECAP *********************************************************************************************
localhost                  : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[ansibleadmin@ip-172-20-10-217 playbooks]$
```

Fuente (Autoría propia)

4.6.4. Construir los playbooks necesarios para el despliegue

En el servidor Ansible controlador se configuran los playbooks necesarios para aprovisionar la máquina nodo una vez el código pase las pruebas de integración. Como se muestra en la figura se configura varios playbooks para cumplir con todas las etapas de despliegue de la aplicación como se muestra en la Figura 29.

Figura 29. *Playbooks utilizados en el pipeline*

```
[ansibleadmin@ip-172-20-10-217 playbooks]$ ll
total 28
-rw-r--r-- 1 root root 822 Jun 29 16:18 descargarDesplegarDocker.yaml
-rw-r--r-- 1 root root 930 Jun 29 22:00 despliegueDockerCompose.yaml
-rw-r--r-- 1 root root 130 Jun 29 15:33 echo.yaml
-rw-r--r-- 1 root root  42 Jun 29 04:53 hosts
-rw-r--r-- 1 root root 195 Jun 29 17:50 installdb.yaml
-rw-r--r-- 1 root root 294 Jun 30 17:39 validarConexionServers.yaml
-rw-r--r-- 1 root root  41 Jun 29 16:13 variables.yaml
[ansibleadmin@ip-172-20-10-217 playbooks]$
```

Fuente (Autoría propia)

Existe inconvenientes al momento de ejecutar comandos de Docker Compose por medio del parámetro Shell en los playbooks, por lo tanto se instala el Módulo community.docker.docker_container que ayuda a gestionar el ciclo de vida de los contenedores Docker, y facilita la construcción de pipeline interactuando directamente con la herramienta Docker Compose como se visualizar en la Figura 30, en la cual se realiza una limpieza de imágenes y se asegura la construcción de los contenedores por medio del archivo de configuración docker-compose.yml (un equivalente a ejecutar el comando Docker-compose up).

Figura 30. *Instrucciones de Ansible para el manejo de Docker Compose*

```
- name: Clean up Docker images
  docker_prune:
   images: yes
   images_filters:
    dangling: false

- name: Make sure compose service is up
  docker_compose:
    project_src: /home/ansibleadmin/lampp-helpact-cicd
    files:
      - "docker-compose.yml"
    state: present
```

Fuente (Autoría propia)

4.6.5. Implementar las medidas de seguridad a nivel de contenedores

Para el despliegue automático de la aplicación mediante contenedores se utiliza Docker Compose que permite conectar todos los servicios que conforman la aplicación y levantarla.

En el archivo Docker-Compose y Dockerfile se definen las configuraciones necesarias y se aplica las medidas de prevención investigadas.

En la configuración se definen tres servicios como se muestra en la Figura 31 y son:

- Servicio Web
- Servicio de base de datos
- Servicio para el gestor de bases de datos PhpMyAdmin

Y de acuerdo a la investigación realizada se sigue las recomendaciones para construir entornos seguros en Docker como se muestra en la Figura 31.

Figura 31. *Implementación y conexión de todos los contenedores*

```
    && apt-get install -y zlib1g-dev \
    && apt-get install -y libonig-dev \
    && rm -rf /var/lib/apt/lists/* \
    && docker-php-ext-install zip

RUN docker-php-ext-install mbstring
RUN docker-php-ext-install zip
RUN docker-php-ext-install gd

RUN a2enmod rewrite
RUN groupadd -r produccion && useradd -r -s /bin/false -g produccion produccion
USER produccion
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  AZURE  JUPYTER

❯ docker-compose up -d

[+] Running 2/3
 ⠿ Container docker-lamp-db-1          Started
 ⠿ Container docker-lamp-phpmyadmin-1  Started
 ⠿ Container docker-lamp-www-1         Starting
```

docker-compose.yml M | Dockerfile M

docker-compose.yml > { } services > { } db > [ ] networks > 0

docker-compose.yml (compose-spec.json)

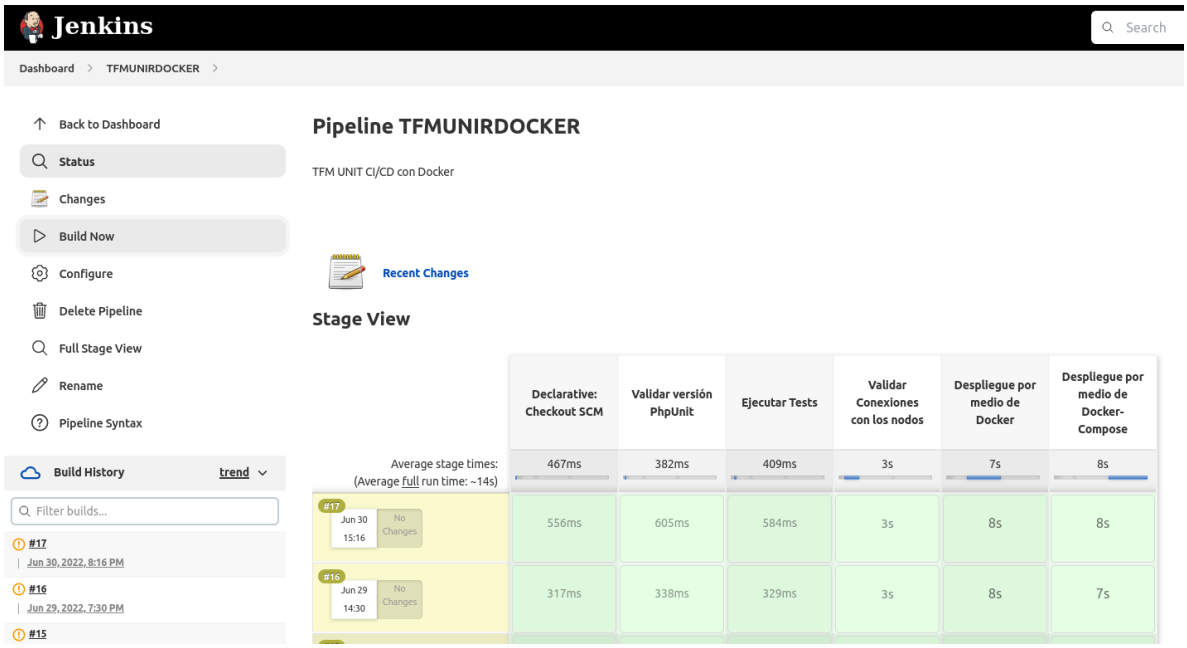
```
version: "3.1"
services:
    db:
        image: mysql
        ports:
            - "3306:3306"

        environment:
            MYSQL_DATABASE: crudemo
            MYSQL_PASSWORD: Tfmunir2022
            MYSQL_ROOT_PASSWORD: Tfmunir2022
        volumes:
            - ./dataBase_crud:/docker-entrypoint-initdb.d
            - ./conf:/etc/mysql/conf.d
            - persistent:/var/lib/mysql
        networks:
            - default
    www:
        build: .
```

Fuente (Autoría propia)

4.6.6. Validar el correcto funcionamiento del pipeline

Una vez terminado todo el pipeline se prueba ejecutando la acción desde Jenkins, que dispone de un panel de control amigable, que brinda información en tiempo real sobre el estado de cada una de las etapas del pipeline que se está ejecutando. Cuando existe algún error el cuadro correspondiente a la etapa se pone de color Rojo, y cuando no existe inconvenientes conserva el color Verde. Como se muestra en la Figura 32 el pipeline ejecuta correctamente cada una de las etapas configuradas.

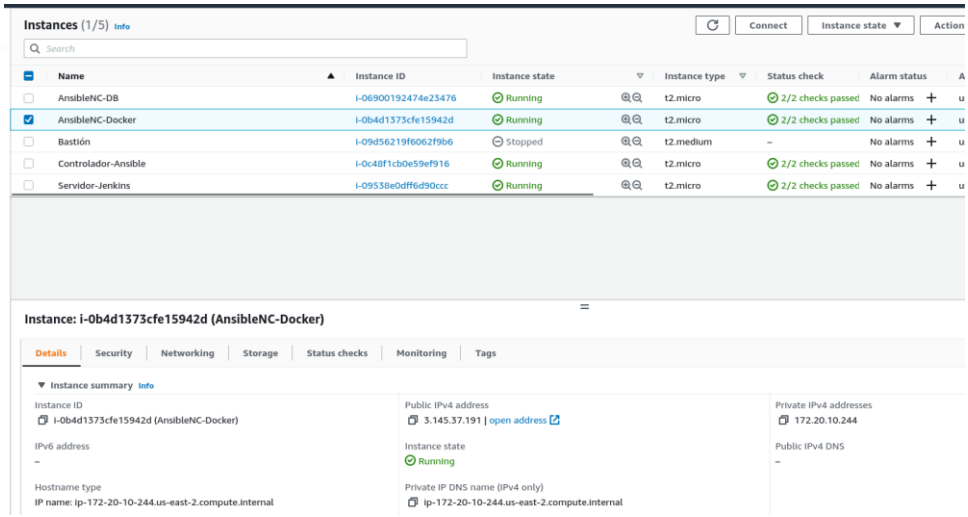
Figura 32. Correcto funcionamiento del pipeline en Jenkins



Fuente (Autoría propia)

Se valida la dirección ip pública del nodo de Docker para comprobar que la aplicación ha sido desplegada. La dirección la obtenemos directamente de la consola de configuración de Aws ya que las ips públicas cambian cada vez que se reinician las instancias como se muestra en la Figura 33.

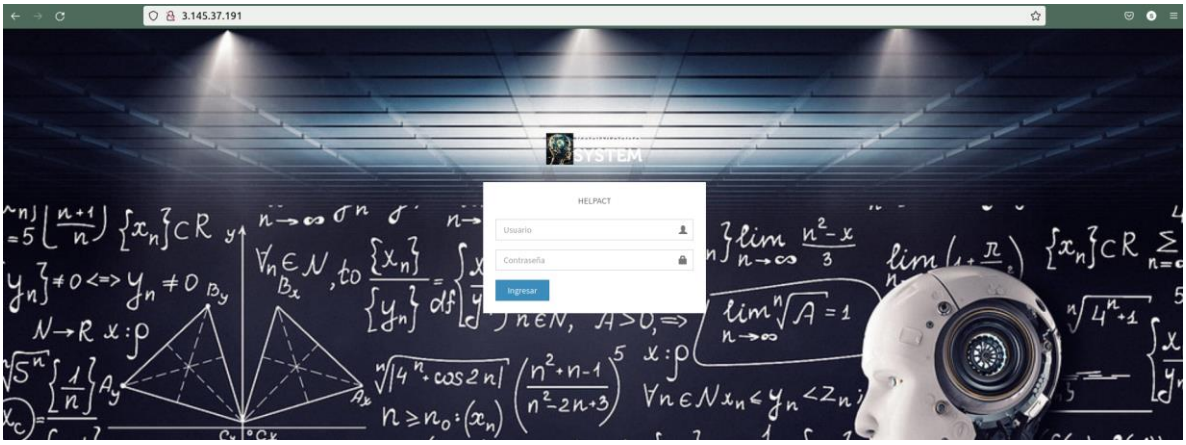
Figura 33. *Consola de Aws para visualizar la ip publica de la instancia EC2*



Fuente (Autoría propia)

Con la ip pública se valida que la aplicación está corriendo correctamente como se muestra en la Figura 34.

Figura 34. *Ventana de login de la aplicación Web*



Fuente (Autoría propia)

Se ingresa a la máquina nodo de Docker y se valida que los servicios están corriendo por medio de comandos de consola de Docker como se muestra en la Figura 35.

Figura 35. *Validar que los contenedores están activos y en estado running*

```
[ansibleadmin@ip-172-20-10-244 lampp-helpact-cicd]$ docker-compose ps
NAME                            COMMAND                  SERVICE             STATUS              PORTS
lampp-helpact-cicd-db-1         "docker-entrypoint.s…"   db                  running             0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp
lampp-helpact-cicd-phpmyadmin-1   "/docker-entrypoint.…"   phpmyadmin          running             0.0.0.0:9000->80/tcp, :::9000->80/tcp
lampp-helpact-cicd-www-1        "docker-php-entrypoi…"   www                 running             0.0.0.0:80->80/tcp, :::80->80/tcp
[ansibleadmin@ip-172-20-10-244 lampp-helpact-cicd]$
```

Fuente (Autoría propia)

4.7. Evaluación

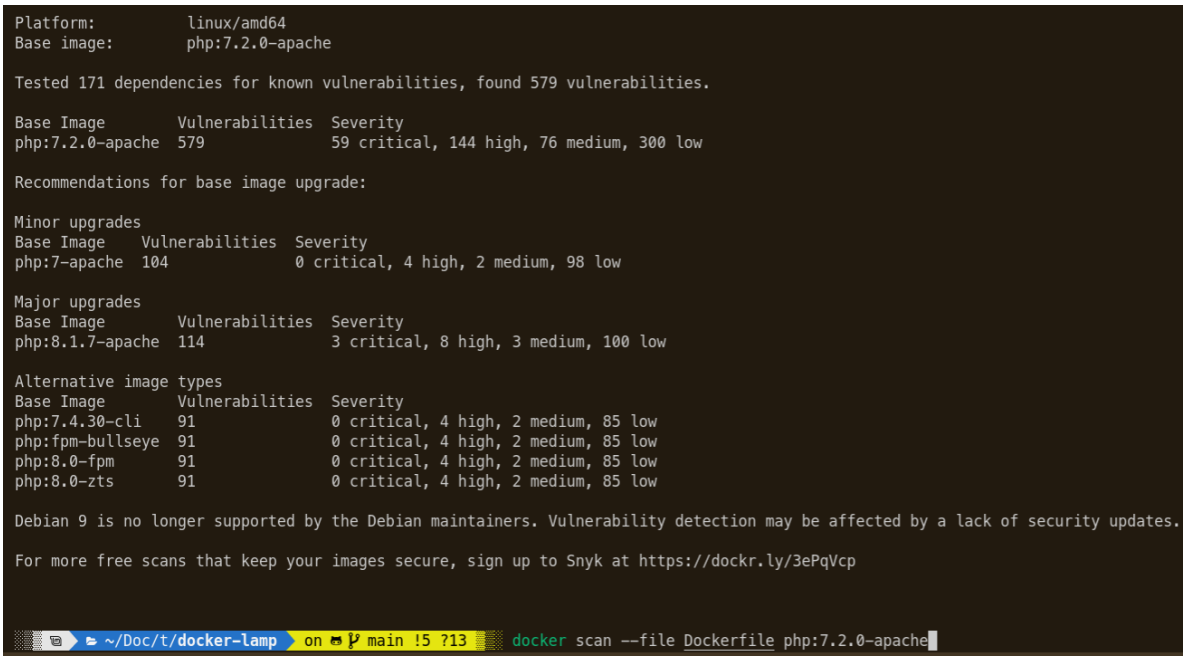
Una vez finalizada la migración de la aplicación a un entorno de nube pública y la implementación de un pipeline para ejecutar un ciclo CI/CD, se procede a realizar una evaluación de su correcto funcionamiento. Para ello se realiza las siguientes acciones:

4.7.1. Evaluar las vulnerabilidades de la imagen generada

Se realiza una evaluación de las vulnerabilidades encontradas en las imágenes mediante la herramienta de Docker Scan.

Se utiliza el comando docker scan --file Dockerfile php:7.2.0-apache arrojando como resultado que la imagen contiene muchas vulnerabilidades en sus dependencias al utilizar la versión 7.2.0 como se muestra en la Figura 36 .

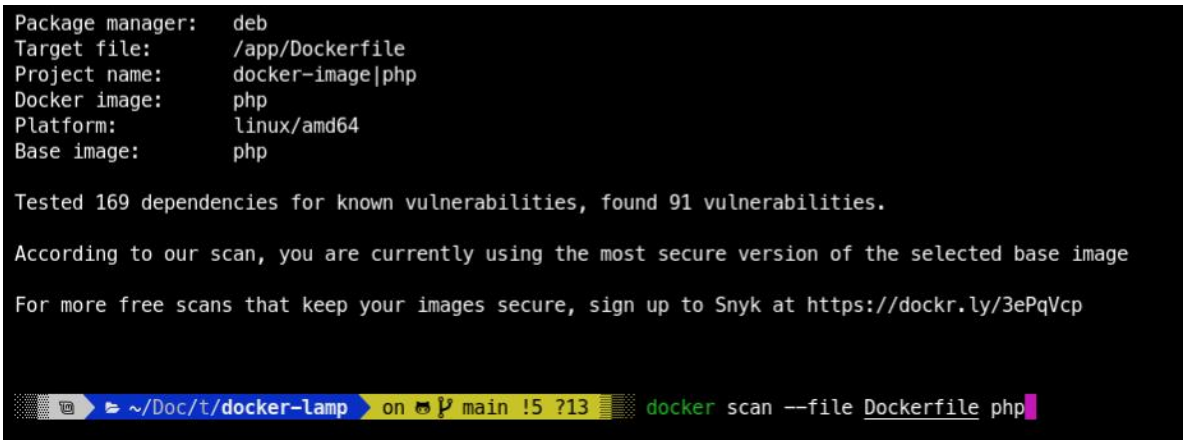
Figura 36. *Vulnerabilidades encontradas con la versión 7.2.0 de la imagen de Php*



Fuente (Autoría propia)

Se procede a utilizar la versión más segura de la imagen base y por medio del comando docker scan --file Dockerfile php se obtiene el siguiente resultado mostrado en la Figura 37, el cual nos informa sobre vulnerabilidades encontradas pero a nivel de dependencias de la aplicación las cuales deben ser evaluadas por el equipo de desarrolladores ya que la imagen ha sido construida mediante el dockerfile siguiendo las recomendaciones investigadas sobre seguridad en los contenedores.

Figura 37. *Informe de vulnerabilidades con la versión más actual de la imagen de Php*

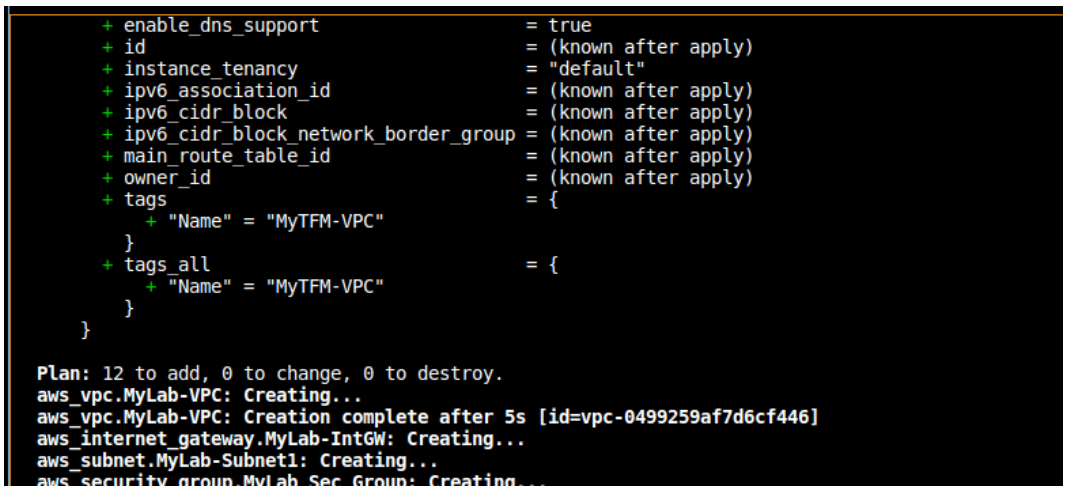


Fuente (Autoría propia)

4.7.2. Evaluar el correcto funcionamiento del despliegue de la infraestructura

Para evaluar el correcto funcionamiento de la infraestructura generada por medio de código se utiliza el comando Terraform Plan como se muestra en la Figura 38, y se revisa todos los parámetros que reflejan por consola y visualizando la arquitectura se valida los cambios. Entonces, una vez generada la infraestructura se realiza las pruebas de ping entre los elementos que conforman la VPC.

Figura 38. *Validación de los cambios a realizar con Terraform*



Fuente (Autoría propia)

En la configuración del archivo de Terraform se especifica el rango de direcciones privadas que se pueden asignar y que corresponden a la red 172.20.10.0/24 de acuerdo a la Tabla 5 :

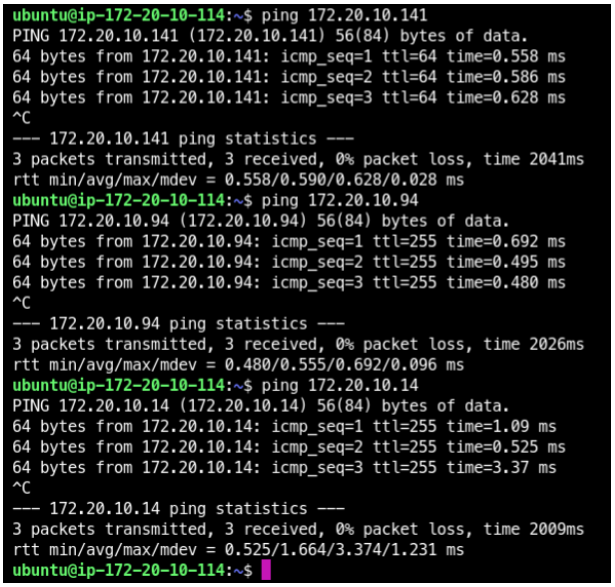
Tabla 5. *Tabla de direcciones ip privadas de las instancias*

Distribución de las Instancias	
Grupo de seguridad 1	**Direcciones IP privadas**
Servidor de Jenkins	172.20.10.141
Servidor Controlador Ansible	172.20.10.94
Servidor Nodo controlado Ansible	172.20.10.14
Grupo de seguridad 2	
Máquina Bastión host	172.20.10.114

Fuente (Autoría propia)

Se ingresa mediante una conexión ssh con la llave pública generada en AWS a la instancia que aloja el host bastión y se procede a realizar pruebas de conectividad con la herramienta ping a cada una de las direcciones ip de las instancias como se muestra en la Figura 39.

Figura 39. Ping a las instancias EC2



Fuente (Autoría propia)

Una vez comprobada la conectividad entre todos los elementos, se procede a validar que el acceso por medio de llaves ssh se realice solamente por medio del host bastión, para lo cual se desactiva la regla de entrada de tráfico que permitía aceptar conexiones desde cualquier ip, reemplazándola por dos reglas ssh las cuales acepten todo el tráfico ssh entre las instancias que conforman el sg1 y el sg2.

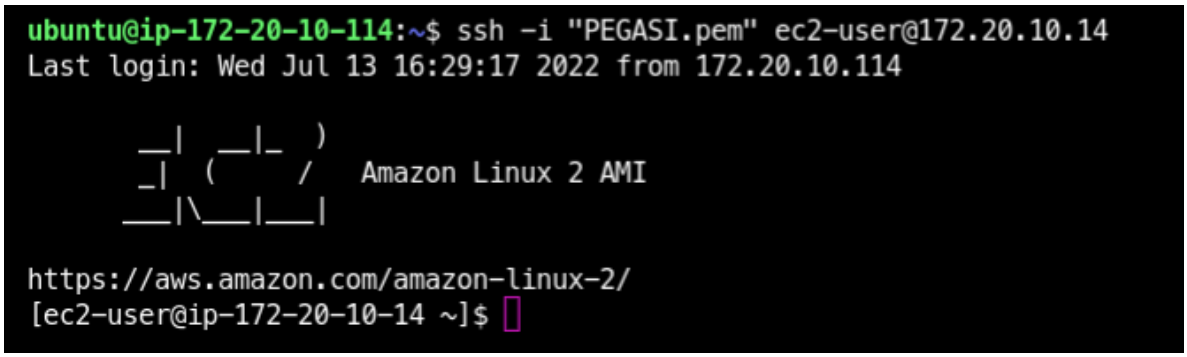
Obteniendo como resultado que la única forma de ingreso y manipulación a las instancias EC2 sea por medio del host bastión.

También es necesario que el host bastión posea la llave pem, por lo tanto se utiliza el siguiente comando para trasferir el archivo a la instancia:

scp -i PEGASI.pem PEGASI.pem ubuntu@54.82.118.219:/home/ubuntu

Y se realiza la prueba de conectividad mediante llaves ssh desde el host bastión a una instancia que aloja el Nodo controlado de Ansible con ip 172.20.10.14 como se muestra en la Figura 40.

Figura 40. *Conexión ssh desde el host bastión*

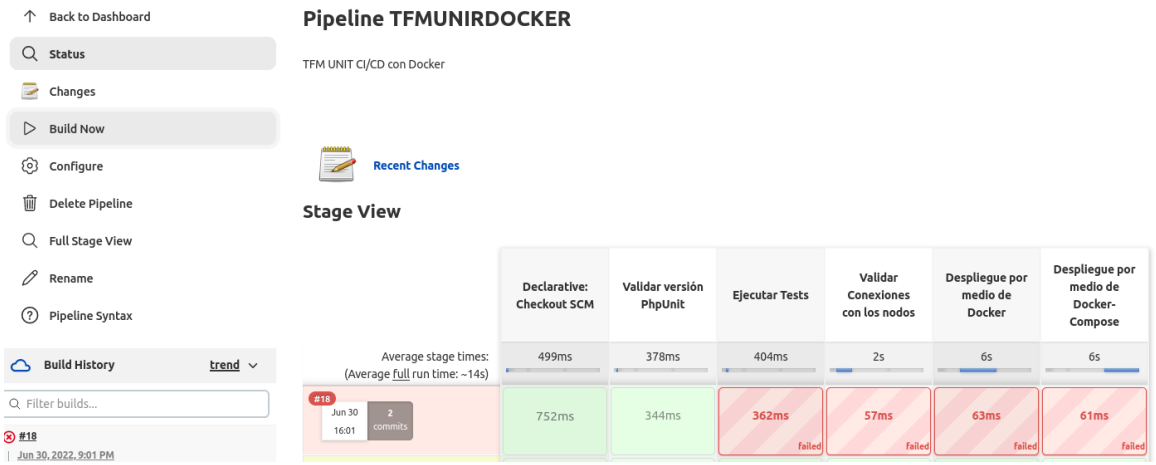


Fuente (Autoría propia)

4.7.3. Evaluar el correcto funcionamiento del pipelice CI/CD

Para evaluar el pipeline se procede a generar un fallo en la prueba de integración para ser exacto se genera un error voluntario en la prueba unitaria. Por lo tanto falla la ejecución del pipeline y se puede validar en los logs de Jenkins como se muestran en las figuras 41 y 42.

Figura 41. *Fallo en las pruebas unitarias en la ventana principal de Jenkins*



Fuente (Autoría propia)

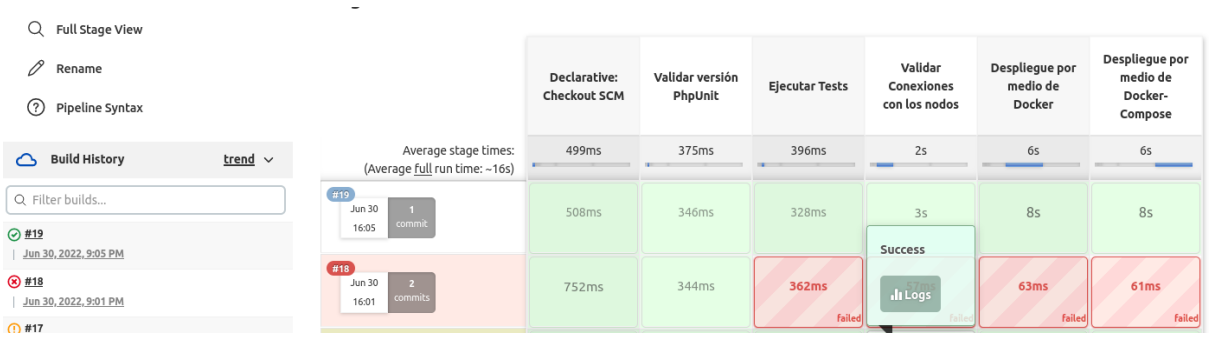
Figura 42. *Log de fallo de las pruebas unitarias*

```
+ ./vendor/bin/phpunit tests
PHPUnit 9.5.21 [44;37m#StandWith[0m[43mUkraine[0m

F......                                                           7 / 7 (100%)

Time: 00:00.055, Memory: 6.00 MB

There was 1 failure:

1) OperationsTest::testAddWithTwoValues
Failed asserting that 7 matches expected 4.

/var/lib/jenkins/workspace/TFMUNIRDOCKER/tests/OperationsTest.php:17
phpvfscomposer:///var/lib/jenkins/workspace/TFMUNIRDOCKER/vendor/phpunit/phpunit/phpunit:97

FAILURES!
Tests: 7, Assertions: 7, Failures: 1.
```

Fuente (Autoría propia)

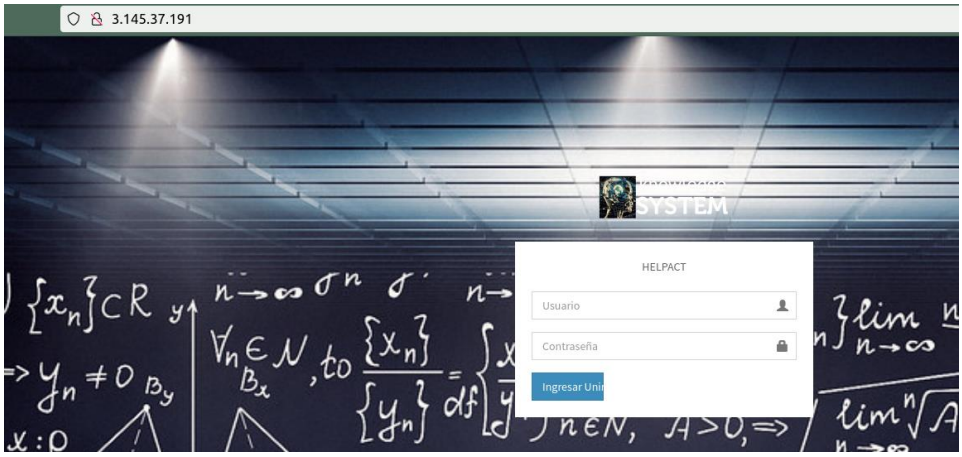
Para validar el funcionamiento del pipeline con respecto al proceso de entrega, se procede a realizar un cambio en el texto de entrada de la aplicación. En el botón utilizado en el login de ingreso se aumenta la palabra Unir y como se muestra en la Figura 43 y Figura 44 se despliega el cambio una vez Jenkins valida que todo el proceso se realizó con éxito.

Figura 43. *Validación de la ejecución del pipeline al realizar un cambio en la aplicación*



Fuente (Autoría propia)

Figura 44. *Validar el cambio en la página principal de la aplicación*



Fuente (Autoría propia)

Cada evaluación realizada ha tenido los resultados esperados y la aplicación ha sido migrada a un entorno de nube pública, se implementó un pipeline para ejecutar un ciclo CI/CD, se construyó los archivos de Docker de acuerdo a las recomendaciones planteadas, de este modo y de acuerdo al análisis de vulnerabilidades se identificó posibles nuevas amenazas que deben ser resueltas, y adicional a esto la infraestructura generada mediante código funciona correctamente de acuerdo al diagrama de arquitectura propuesto.

5. Conclusiones y trabajo futuro

5.1.Conclusiones

Basado en los objetivos planteados y una vez finalizado la migración de la aplicación Web a un entorno de nube pública implementando un ciclo CI/CD con el marco de trabajo DevOps teniendo un énfasis a la seguridad de los contenedores, se llega a las siguientes conclusiones.

- El presente trabajo describe la forma de migrar una aplicación Web pequeña e implementar un ciclo de integración y entrega continua, a través de la utilización de herramientas tecnológicas y una filosofía de trabajo DevOps.
- En el proceso de implementación del pipeline se tomó en cuenta las fases más importantes y básicas que debe tener un proyecto. Por lo tanto se implementó las pruebas unitarias y el despliegue mediante contenedores que facilita la entrega ágil de productos.
- Se identifica un grupo de herramientas tales como Vagrant, Terraform, Jenkins, Ansible y Docker que interactúan entre ellas, permitiendo implementar un ciclo CI/CD, además de contar con documentación detallada y abundante en el internet sobre su funcionamiento.
- Mediante la investigación realizada se implementan las recomendaciones sobre materia de seguridad, que se basan en tener buenas prácticas al momento de generar las imágenes de Docker por medio de un archivo dockerfile.
- La arquitectura de red propuesta e implementada en el proveedor de nube Amazon Web Services permitió migrar la aplicación e implementar el pipeline.
- La infraestructura creada y el ciclo CI/CD funcionó correctamente, así como las herramientas las cuales no registraron ningún inconveniente, y todos los problemas de configuración se solventaron por medio de la investigación tomando como referencia la documentación oficial de cada herramienta, ya que tienen muchos años en la industria y se utilizan en grandes proyectos .
- Utilizar un marco de trabajo DevOps con metodologías ágiles permitió dividir el proyecto en pequeñas piezas o iteraciones incrementales y a pesar de los cambios

que se realizaron a lo largo del desarrollo del trabajo, obtener un producto final con todas sus funcionalidades y cumplir con los objetivos planteados.

- Mediante la utilización de la herramienta de escaneo de imágenes de Docker se detecta que existen muchas vulnerabilidades a nivel de las dependencias propias de la aplicación pero no a nivel de la imagen base.

5.2. Líneas de trabajo futuro

Luego de alcanzar los objetivos propuestos se identifican líneas de trabajo a futuro para dar continuidad al presente proyecto y se describen como:

1. Integrar más tipos de pruebas al pipeline tales como pruebas de rendimiento o las pruebas de la interfaz de usuario a la aplicación.
2. El código tienen pocas pruebas unitarias, por lo tanto, es necesario aumentarlas de tal manera que cubra la mayor parte del código de la aplicación
3. Implementar una plataforma para el análisis de código estático que ayude a mejorar la calidad del código de la aplicación y se integre al pipeline ya establecido.
4. Implementar herramientas de monitoreo de la aplicación que permitan el procesamiento y análisis de logs de forma automatizada, agregando métricas tales como la Información sobre el estado de la instancia para validar el consumo de los recursos, medir el tráfico de red del servidor y monitorear el estado de la memoria.
5. Implementar una herramienta de visualización de los datos recolectados tal como Grafana lo cual permitirá presentar los datos de monitorización de una forma agradable y de fácil entendimiento para el equipo DevOps.
6. Agregar métricas de rendimiento en la aplicación para medir los tiempos de respuesta de las solicitudes, y poder detectar anomalías en su funcionamiento que estén provocando tiempos de espera excesivos al momento de ingresar a la plataforma o cuando se utilice todas sus funciones.
7. Realizar pruebas de penetración o pen test a la infraestructura y la aplicación para identificar y probar vulnerabilidades de seguridad, de tal modo que se mejore las configuraciones de ser necesario y se identifiquen nuevas amenazas.

8. Agregar servicios de AWS para mejorar la seguridad como Amazon inspector que es un escáner de vulnerabilidades de mínimo impacto, que reúne información sobre los procesos o la red donde están alojadas las instancias, y entrega un informe con una lista de problemas detectados. De este modo se agrega una capa más de seguridad a la solución propuesta.

9. El equipo de desarrolladores de la aplicación debe trabajar en la actualización de la aplicación a la versión 8 de Php con sus respectivas dependencias, ya que se detectó por medio de la utilización de la herramienta de escaneo de vulnerabilidades que existen muchas vulnerabilidades en la versión 7.

6. REFERENCIAS BIBLIOGRÁFICAS

AcensTechnologies. (2014). *AcensTechnologies.* Obtenido de AcensTechnologies: https://www.acens.com/wp-content/images/2014/12/wp-composer-acens.pdf

Agarwal, G. (2021). *Modern DevOps Practices.* Packt Publishing Ltd.

Antony, M. (2018). Docker ecosystem - Vulnerability Analysis.

AWS. (2022). *AWS*. Obtenido de AWS: https://aws.amazon.com/es/what-is-aws/

AWS. (2022). *AWS*. Obtenido de https://aws.amazon.com/es/docker/

Brady, K., Moon, S., Nguyen Tuan, & Coffman, J. (s.f.). Docker Container Security in Cloud Computing.

Davis, J., & Daniels, R. (2016). *Effective DevOps.* Sebastopol: O'Reilly Media.

Digite. (2022). *Digite*. Obtenido de Digite: https://www.digite.com/es/agile/pruebas-unitarias/

Docker. (2021). *Docker*. Obtenido de Docker: https://docs.docker.com/engine/install/linux-postinstall/

Encora. (2020). *Encora*. Obtenido de Encora: https://www.encora.com/es/blog/construyendo-imagenes-seguras-con-docker

Google Cloud. (2018). *cloud google.* Obtenido de https://cloud.google.com/architecture/best-practices-for-building-containers

Márquez, E. (2022). *concurrencylabs*. Obtenido de concurrencylabs: https://www.concurrencylabs.com/blog/choose-your-aws-region-wisely/

RedHat. (2020). *RedHat*. Obtenido de RedHat: https://www.redhat.com/es/topics/containers/containers-vs-vms

Rice, L. (2020). *Container Security.* O'Reilly Media, Inc.

Rotibi Bola, H. C. (2021). *Containers and Kubernetes.* Obtenido de Redhat: https://www.redhat.com/

shah, j., Dubaria, D., & Widhalm, J. (s.f.). A Survey of DevOps tools for Networking. *IEEE*.

Soni, M. (2016). *DevOps for Web Development.* Packt Publishing.

Sujay Vailshery, L. (2020). *statista*. Obtenido de https://www.statista.com/statistics/1104543/worldwide-container-technology-use/

Wetter, D. (2020). *OWASP Docker Top 10.* OWASP .

Wetter, D. (29 de 12 de 2022). *owasp.* Obtenido de owasp: https://owasp.org/www-project-docker-top-10/

Anexo A. Archivo de configuración de Jenkins

```
pipeline{
    //Directivas
    agent any
    stages {
        // Definir las etapas del pipeline
        // stage 1. Validar la version de PHP UNIT y si esta instalado
        stage ('Validar versión PhpUnit'){
            steps {
                sh 'phpunit --version'
            }
        }
        // Stage2 : realizar el test en base al archivo xml
        stage ('Ejecutar Tests'){
            steps {
                sh './vendor/bin/phpunit tests'
            }
        }
        // Stage 3 : Validar conexión con los servidores
        stage ('Validar Conexiones con los nodos'){
            steps {
                echo "Validar conexión con los servidores...."
                sshPublisher(publishers:
                [sshPublisherDesc(
                    configName: 'Controlador_Ansible',
```

```
            transfers: [
               sshTransfer(
                     cleanRemote:false,
                     execCommand: 'ansible-playbook
/opt/playbooks/validarConexionServers.yaml -i /opt/playbooks/hosts',
                     execTimeout: 120000
               )
            ],
            usePromotionTimestamp: false,
            useWorkspaceInPromotion: false,
            verbose: false)
            ])
      }
   }
   // Stage 4 : Desplegar los cambios en el Contenedor Docker
   stage ('Despliegue por medio de Docker'){
      steps {
         echo "Despliegue ...."
         sshPublisher(publishers:
         [sshPublisherDesc(
            configName: 'Controlador_Ansible',
            transfers: [
               sshTransfer(
                     cleanRemote:false,
```

```
                    execCommand: 'ansible-playbook
/opt/playbooks/descargarDesplegarDocker.yaml -i /opt/playbooks/hosts',
                    execTimeout: 120000
                )
            ],
            usePromotionTimestamp: false,
            useWorkspaceInPromotion: false,
            verbose: false)
            ])
        }
    }
    // Stage 4 : Desplegar los cambios con Docker-Compose
    stage ('Despliegue por medio de Docker-Compose'){
        steps {
            echo "Despliegue ...."
            sshPublisher(publishers:
            [sshPublisherDesc(
                configName: 'Controlador_Ansible',
                transfers: [
                    sshTransfer(
                        cleanRemote:false,
                        execCommand: 'ansible-playbook
/opt/playbooks/desplieguedockercompose.yaml -i /opt/playbooks/hosts',
                        execTimeout: 120000
                    )
```

```
            ],
            usePromotionTimestamp: false,
            useWorkspaceInPromotion: false,
            verbose: false)
            ])
      }
   }
   // Stage 4 : Desplegar los cambios con Docker-Compose
   stage ('Despliegue por medio de Docker-Compose2'){
      steps {
         echo "Despliegue ...."
         sshPublisher(publishers:
         [sshPublisherDesc(
            configName: 'Controlador_Ansible',
            transfers: [
               sshTransfer(
                     cleanRemote:false,
                     execCommand: 'ansible-playbook /opt/playbooks/composeonly.yaml -i
/opt/playbooks/hosts',
                     execTimeout: 12000000
               )
            ],
            usePromotionTimestamp: false,
            useWorkspaceInPromotion: false,
            verbose: false
```

Anexo B. Consola de salida de Jenkins

Started by user pegasi

Obtained Jenkinsfile from git https://github.com/SantinoSuntaxi/unitTest.git

[Pipeline] Start of Pipeline

[Pipeline] node

Running on Jenkins in /var/lib/jenkins/workspace/tfmUnir

[Pipeline] {

[Pipeline] stage

[Pipeline] { (Declarative: Checkout SCM)

[Pipeline] checkout

The recommended git tool is: git

No credentials specified

> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/tfmUnir/.git # timeout=10

Fetching changes from the remote Git repository

> git config remote.origin.url https://github.com/SantinoSuntaxi/unitTest.git # timeout=10

Fetching upstream changes from https://github.com/SantinoSuntaxi/unitTest.git

> git --version # timeout=10

> git --version # 'git version 2.25.1'

> git fetch --tags --force --progress -- https://github.com/SantinoSuntaxi/unitTest.git +refs/heads/*:refs/remotes/origin/* # timeout=10

> git rev-parse refs/remotes/origin/main^{commit} # timeout=10

Checking out Revision 485c782c5085caf181b35f017bb6bbe78b38c2cf (refs/remotes/origin/main)

> git config core.sparsecheckout # timeout=10

> git checkout -f 485c782c5085caf181b35f017bb6bbe78b38c2cf # timeout=10

Commit message: "Agregar una etapa con docker compose python"

> git rev-list --no-walk 485c782c5085caf181b35f017bb6bbe78b38c2cf # timeout=10

[Pipeline] }

[Pipeline] // stage

[Pipeline] withEnv

[Pipeline] {

[Pipeline] stage

[Pipeline] { (Validar versión PhpUnit)

[Pipeline] sh

+ phpunit --version

PHPUnit 8.5.2 by Sebastian Bergmann and contributors.

[Pipeline] }

[Pipeline] // stage

[Pipeline] stage

[Pipeline] { (Ejecutar Tests)

[Pipeline] sh

+ ./vendor/bin/phpunit tests

PHPUnit 9.5.21 [44;37m#StandWith[0m[43mUkraine[0m

..... 7 / 7 (100%)

Time: 00:00.005, Memory: 6.00 MB

OK (7 tests, 7 assertions)

[Pipeline] }

[Pipeline] // stage

[Pipeline] stage

[Pipeline] { (Validar Conexiones con los nodos)

[Pipeline] echo

Validar conexión con los servidores ....

[Pipeline] sshPublisher

SSH: Connecting from host [ip-172-20-10-141]

SSH: Connecting with configuration [Controlador_Ansible] ...

SSH: EXEC: completed after 3,409 ms

SSH: Disconnecting configuration [Controlador_Ansible] ...

SSH: Transferred 0 file(s)

[Pipeline] }

[Pipeline] // stage

[Pipeline] stage

[Pipeline] { (Despliegue por medio de Docker)

Despliegue ....

[Pipeline] sshPublisher

SSH: Connecting from host [ip-172-20-10-141]

SSH: Connecting with configuration [Controlador_Ansible] ...

SSH: EXEC: completed after 9,211 ms

SSH: Disconnecting configuration [Controlador_Ansible] ...

SSH: Transferred 0 file(s)

[Pipeline] }

[Pipeline] // stage

[Pipeline] stage

[Pipeline] { (Despliegue por medio de Docker-Compose)

Despliegue ....

[Pipeline] sshPublisher

SSH: Connecting from host [ip-172-20-10-141]

SSH: Connecting with configuration [Controlador_Ansible] ...

SSH: EXEC: completed after 8,815 ms

SSH: Disconnecting configuration [Controlador_Ansible] ...

SSH: Transferred 0 file(s)

[Pipeline] }

[Pipeline] // stage

[Pipeline] stage

[Pipeline] { (Despliegue por medio de Docker-Compose2)

[Pipeline] echo

Despliegue ....

[Pipeline] sshPublisher

SSH: Connecting from host [ip-172-20-10-141]

SSH: Connecting with configuration [Controlador_Ansible] ...

SSH: EXEC: completed after 255,595 ms

SSH: Disconnecting configuration [Controlador_Ansible] ...

SSH: Transferred 0 file(s)

[Pipeline] }

[Pipeline] // stage

[Pipeline] }

[Pipeline] // withEnv

[Pipeline] }

[Pipeline] // node

[Pipeline] End of Pipeline

Finished: SUCCESS

Anexo C. Informe de vulnerabilidad realizado a la imagen

docker scan --file Dockerfile php

Testing php...

✗ Low severity vulnerability found in util-linux/libblkid1

Description: Information Exposure

Info: https://snyk.io/vuln/SNYK-DEBIAN11-UTILLINUX-2401081

Introduced through: e2fsprogs@1.46.2-2, util-linux/mount@2.36.1-8+deb11u1, pkg-config@0.29.2-1, util-linux/libuuid1@2.36.1-8+deb11u1, util-linux@2.36.1-8+deb11u1, util-linux/bsdutils@1:2.36.1-8+deb11u1, util-linux/libsmartcols1@2.36.1-8+deb11u1

From: e2fsprogs@1.46.2-2 > util-linux/libblkid1@2.36.1-8+deb11u1

From: util-linux/mount@2.36.1-8+deb11u1 > util-linux/libblkid1@2.36.1-8+deb11u1

From: util-linux/mount@2.36.1-8+deb11u1 > util-linux@2.36.1-8+deb11u1 > util-linux/libblkid1@2.36.1-8+deb11u1

and 14 more...

Image layer: Introduced by your base image (php)

✗ Low severity vulnerability found in krb5/libk5crypto3

Description: CVE-2004-0971

Info: https://snyk.io/vuln/SNYK-DEBIAN11-KRB5-519904

Introduced through: krb5/libk5crypto3@1.18.3-6+deb11u1, curl@7.74.0-1.3+deb11u1, krb5/libkrb5-3@1.18.3-6+deb11u1, krb5/libgssapi-krb5-2@1.18.3-6+deb11u1, gcc-defaults/g++@4:10.2.1-1, meta-common-packages@meta

From: krb5/libk5crypto3@1.18.3-6+deb11u1

From: curl@7.74.0-1.3+deb11u1 > curl/libcurl4@7.74.0-1.3+deb11u1 > krb5/libgssapi-krb5-2@1.18.3-6+deb11u1 > krb5/libk5crypto3@1.18.3-6+deb11u1

From: curl@7.74.0-1.3+deb11u1 > curl/libcurl4@7.74.0-1.3+deb11u1 > krb5/libgssapi-krb5-2@1.18.3-6+deb11u1 > krb5/libkrb5-3@1.18.3-6+deb11u1 > krb5/libk5crypto3@1.18.3-6+deb11u1

and 6 more...

Image layer: Introduced by your base image (php)

✗ High severity vulnerability found in curl/libcurl4

Description: Improper Certificate Validation

Info: https://snyk.io/vuln/SNYK-DEBIAN11-CURL-2813769

Introduced through: curl/libcurl4@7.74.0-1.3+deb11u1, curl@7.74.0-1.3+deb11u1

From: curl/libcurl4@7.74.0-1.3+deb11u1

From: curl@7.74.0-1.3+deb11u1 > curl/libcurl4@7.74.0-1.3+deb11u1

From: curl@7.74.0-1.3+deb11u1

Image layer: Introduced by your base image (php)

Package manager: deb

Target file: /app/Dockerfile

Project name: docker-image|php

Docker image: php

Platform: linux/amd64

Base image: php

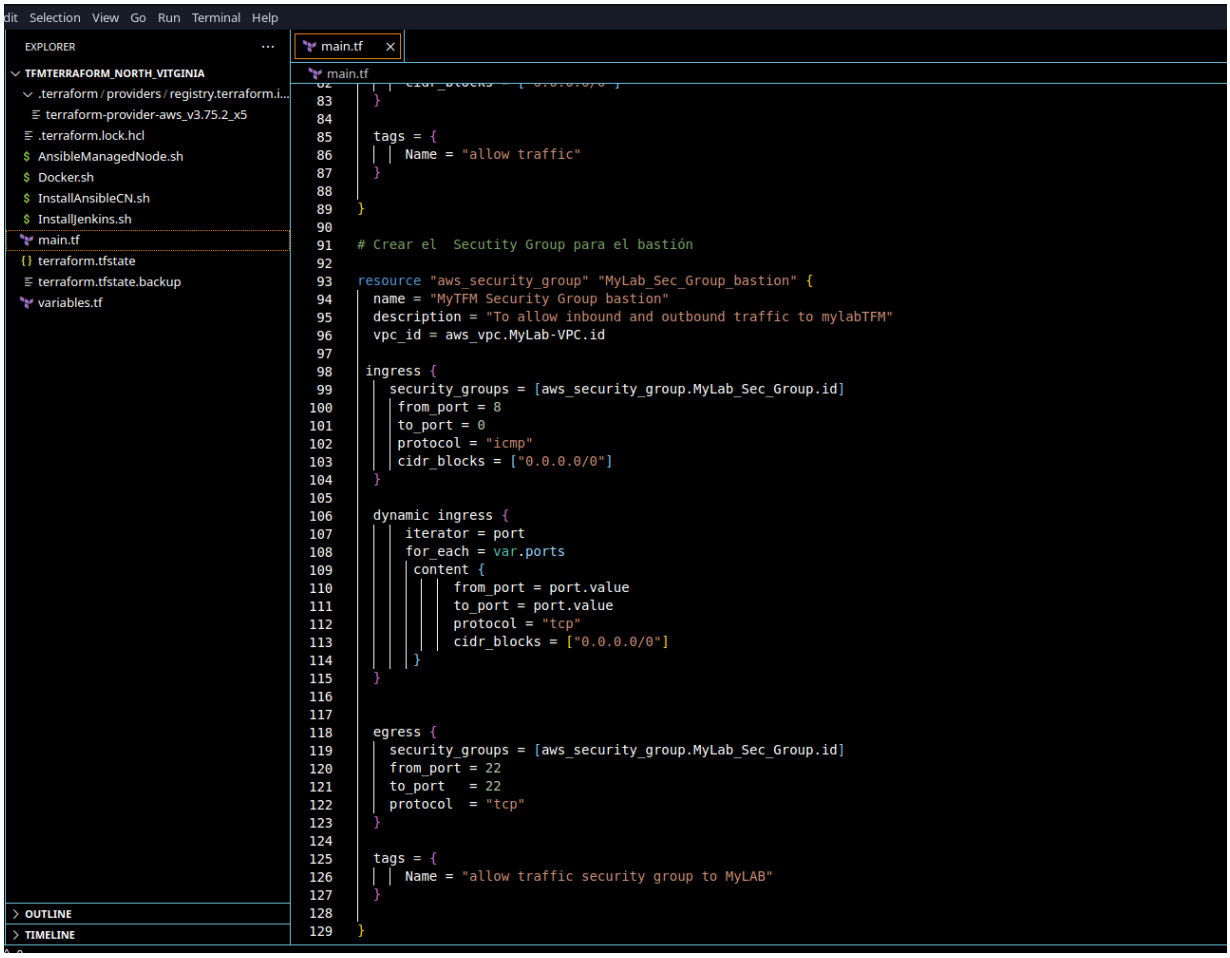
Tested 169 dependencies for known vulnerabilities, found 91 vulnerabilities.

According to our scan, you are currently using the most secure version of the selected base image

For more free scans that keep your images secure, sign up to Snyk at https://dockr.ly/3ePqVcp

Anexo D. Construcción de la infraestructura con Terraform

Figura 45. *Archivo de configuracion de Terraform y los escripts utilizados*



Fuente (Autoría propia)