Universidad Nacional del Nordeste

Facultad de Ciencias Exactas y Naturales y Agrimensura

Especialización en Tecnologías de la Información

Trabajo Final

“Guía y repositorio para el despliegue de aplicaciones Web en servidores Linux utilizando contenedores”

Autor: Jesús Andrés Zini

Director: Emanuel Irrazábal

Año 2023

Dedicado a:

Dedicado a mi profesor orientador, Emanuel, cuya asistencia y acompañamiento fueron fundamentales en el desarrollo de este trabajo final de especialización. A mi amigo Oscar, que fue quien me convenció y motivó a estudiar esta especialización. Además, dedico este trabajo también a mi novia Norma, quien ha sido mi mayor apoyo a lo largo de todo el proceso brindándome su compañía y aliento.

Resumen

El objetivo principal del trabajo es estandarizar la forma de desplegar aplicaciones web monolíticas en servidores privados virtuales para equipos pequeños de desarrollo de software.

Aquí se propone seguir un método similar al utilizado por equipos de desarrollo de aplicaciones web distribuidas y de gran tamaño, que emplean una variedad de recursos, tales como microservicios, múltiples servicios de computación en la nube, herramientas de orquestación de contenedores, múltiples instancias de la aplicación, entre otros. Este método se trata de la utilización de un repositorio específico para documentar y almacenar archivos relacionados al despliegue de la aplicación.

Se construye de manera iterativa e incremental un repositorio base con archivos de configuración de contenedores y scripts de despliegue, para el despliegue de aplicaciones web monolíticas.

El procedimiento desarrollado es verificado y mejorado mediante su implementación en un equipo de trabajo en una empresa de la industria del software regional. Se utiliza un modelo de ciclo de vida evolutivo para guiar el desarrollo de la metodología y adaptarla según las necesidades específicas del equipo y las aplicaciones web en cuestión.

Agradecimientos

Este trabajo se ha llevado a cabo en el marco de la empresa Devlights, y quiero expresar mi agradecimiento a todos sus integrantes y compañeros de trabajo que han contribuido de diversas formas. Agradezco especialmente a aquellos que generosamente dedicaron su tiempo para responder los cuestionarios y utilizaron la solución propuesta en este trabajo.

Índice de Contenidos

Índice de figuras

Índice de tablas

Glosario

Aplicación web monolítica: aplicación web construida como una sola entidad.

Backend: aplicación servidor que procesa y almacena datos.

Bash Script: secuencia de comandos para automatizar tareas en sistemas operativos Unix.

CI/CD: acrónimo de Integración Continua e Implementación Continua. Es una práctica de desarrollo de software que se centra en automatizar y agilizar el proceso de entrega de software.

Docker: plataforma para empaquetar y ejecutar aplicaciones en diferentes entornos.

Docker Registry: repositorio para almacenar y distribuir imágenes de Docker.

Frontend: aplicación cliente que interactúa con el usuario.

Git: herramienta de control de versiones para el desarrollo colaborativo de software.

IaC: Infraestructura como código, enfoque para gestionar la infraestructura de TI como si fuera software.

Microservicios: enfoque arquitectónico y organizativo que utiliza pequeños servicios independientes para la construcción de una aplicación software.

Pipeline: secuencia de procesamiento de datos en la que la salida de cada etapa se utiliza como entrada para la siguiente etapa.

Serverless: modelo de computación en la nube que permite a los desarrolladores crear y ejecutar aplicaciones sin administrar la infraestructura subyacente.

SSL: abreviatura de *Secure Sockets Layer* (capa de sockets seguros), un protocolo para navegadores y servidores web que permite autenticar, cifrar y descifrar la información enviada a través de Internet.

VPS: abreviatura de *Virtual Private Server* (servidor privado virtual), es una máquina que aloja todo el software y los datos necesarios para ejecutar una aplicación o un sitio web.

1. Introducción

Este capítulo es una introducción a la investigación que se llevará a cabo en este trabajo. Se presentará la fundamentación del presente trabajo final integrador, así como los objetivos generales y específicos del mismo. Además, se discutirán los desafíos asociados con el despliegue de aplicaciones web y se presentarán las soluciones propuestas en este trabajo.

1.1 Fundamentación

En equipos de desarrollo de software de empresas pequeñas o medianas, uno de los procesos clave es el despliegue a Internet de las aplicaciones web, para que las mismas puedan ser accedidas por los miembros del equipo o los clientes [1]; tanto para entornos de desarrollo y pruebas, como también hacia entornos de producción.

En muchas empresas no se despliega el software adecuadamente, lo que puede ocasionar inconvenientes y retrasos. Un problema frecuente radica en que algunos desarrolladores, si bien cumplen con su función de construcción del código fuente, cuando llega la hora de desplegar la aplicación, carecen de los conocimientos necesarios para completar esta tarea. Esto genera problemas en la integración continua de lo desarrollado, y representa una barrera en su trabajo, debido a la dependencia de un administrador de sistemas que los guíe en el proceso o que lo suplante en la tarea. Como bien se explican en el libro “Site Reliability Engineer” [2].

> *El rol de administrador de sistemas requiere un conjunto de habilidades marcadamente diferente al que se requiere de los desarrolladores de un producto, los desarrolladores y los administradores de sistemas se dividen en equipos discretos: ‘desarrollo’ y ‘operaciones’.*

En empresas pequeñas o compuestas por pequeños equipos de trabajo tal y como sucede en la región nordeste de Argentina no es económicamente viable contratar personal con habilidades específicas para una sola tarea o rol. En estos casos, una de las soluciones más utilizadas es contar con equipos multidisciplinarios, que puedan abordar distintas tareas de manera eficiente y colaborativa [3].

Para tener un conocimiento más detallado de estos aspectos se realizó una breve encuesta a ocho equipos de tres empresas de la región nordeste de Argentina acerca de los problemas que enfrentan en relación con el despliegue de aplicaciones web. Los hallazgos luego del procesamiento de las respuestas dan cuenta de los siguientes interrogantes o inquietudes que un equipo de trabajo enfrenta a diario:

- ¿Dónde y cómo desplegar el proyecto?
- ¿Cómo configurar el servidor con todas las herramientas que el proyecto necesita?
- ¿Cómo agregar seguridad en la capa de transporte (SSL por sus siglas en inglés) a la aplicación web cliente (interfaz de usuario)?
- ¿Cómo desplegar una nueva versión de la aplicación después de desarrollar actualizaciones para la misma?

De acuerdo a lo indicado por los equipos de desarrollo de software que participaron en la encuesta, esto es un problema por varios motivos, como, por ejemplo:

- El proceso de despliegue de una aplicación web, ya sea para su lanzamiento inicial o para desplegar una nueva versión, a menudo se ve afectado por un aumento significativo en el tiempo necesario para completar el proceso, en promedio de dos a tres horas.
- Se precisa la ayuda y asesoramiento de otros desarrolladores con experiencia previa o conocimiento de la materia; lo que puede generar una dependencia en su disponibilidad para poder llevar a cabo el proceso de despliegue.
- Es común recurrir a alternativas de despliegue que no son las opciones más adecuadas dentro del contexto en el que se encuentran, lo cual resulta en un aumento de costes y en una oportunidad perdida para aprovechar los recursos ya existentes en la empresa, como ser un servidor privado virtual que ya se encuentra en funcionamiento.

1.1.1 Despliegue de aplicaciones web realizado de forma tradicional.

Un enfoque clásico para el despliegue de los proyectos, pero todavía ampliamente utilizado herramientas de control de versiones como, por ejemplo, Git [4], para luego configurarlo y ejecutarlo. Esto se representa en la Fig. 1.

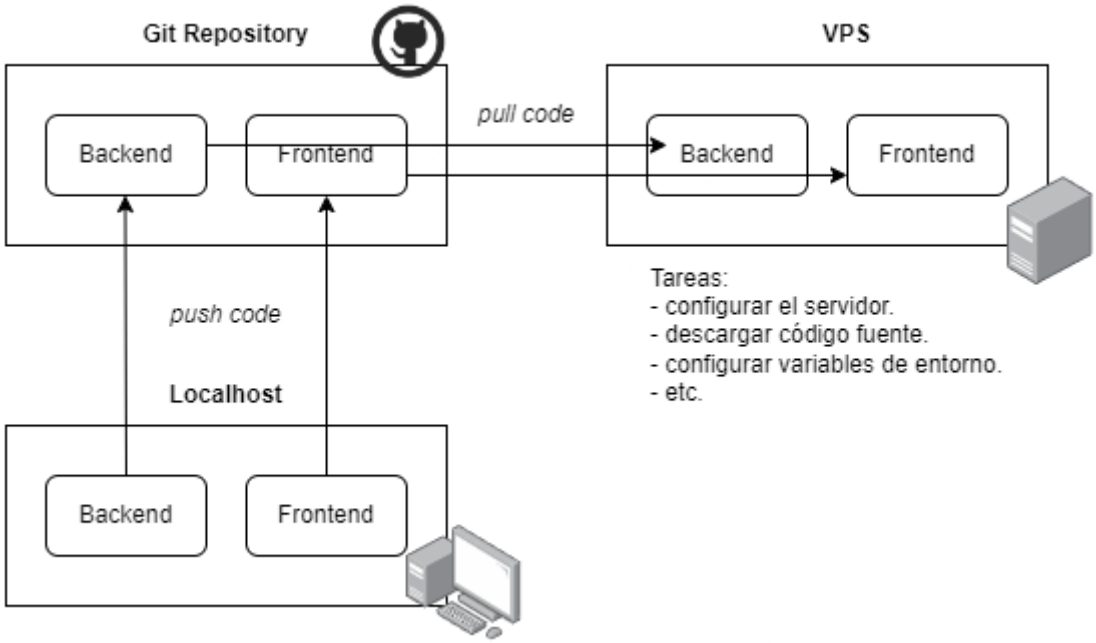


Fig. 1*: Enfoque clásico de despliegue de aplicaciones web monolíticas a VPS.*

Si bien este método funciona, no podemos ignorar que se está desaprovechando la existencia de herramientas más modernas como ser los contenedores, para aprovechar mejor los recursos y automatizar el proceso. De acuerdo con [5], algunas desventajas que esto trae consigo son:

- Se deben instalar los programas y las dependencias necesarias para la ejecución del proyecto en el servidor.
- Se debe subir el código fuente de la aplicación en su totalidad al servidor.
- Se deben realizar las configuraciones del proyecto una vez el código esté en el servidor, como ser agregar las variables de entorno, entre otras.
- Resulta difícil de migrar el proyector a otro servidor, porque se debe realizar todo el proceso nuevamente.
- Se pierde la pista de los programas y versiones de los mismos instalados en el servidor.
- Desplegar más de una aplicación en el mismo servidor puede generar incompatibilidades: diferentes versiones del entorno de ejecución, asignación o utilización de puertos para cada aplicación, por ejemplo.

1.1.1 Despliegue de aplicaciones web utilizando contenedores.

El despliegue de aplicaciones web requiere un entorno de tiempo de ejecución que incluya el servidor web, los entornos de ejecución del lenguaje, las bases de datos y otros componentes, pero no requiere un sistema operativo completo para funcionar. Una solución actual a este problema es la construcción de este conjunto de elementos software unidos conteniendo todos los elementos necesarios para la ejecución de la aplicación contenida. Existen diferentes herramientas que posibilitan la construcción de estos contenedores. Una de las más conocidas es Docker, un proyecto de código abierto sobre plataforma Linux creada en marzo del año 2013 [6].

Al usar Docker, podemos empaquetar una aplicación web y su entorno de ejecución en "imágenes", para así desarrollarla, probarla, y desplegarla en cualquier sistema o servidor. Además, los contenedores Docker pueden iniciarse más rápidamente y utilizar los recursos de manera más eficiente que las máquinas virtuales [7].

Entonces, la utilización de contenedores puede ser útil para resolver algunos de los problemas anteriormente mencionados, porque se empaqueta el software en unidades estandarizadas para su fácil desarrollo, transporte y despliegue [1]. Por lo cual no es necesario la instalación de programas y dependencias para el proyecto en el servidor. Como se menciona en [6].

> *La portabilidad y reproducibilidad de un proceso/programa en contenedores brinda la oportunidad de mover y escalar nuestras aplicaciones a través de nubes y centros de datos. Los contenedores garantizan de manera efectiva que esas aplicaciones se ejecuten de la misma manera en cualquier lugar, lo que nos permite aprovechar rápida y fácilmente todos estos entornos.*

Sin embargo, la utilización de contenedores en el servidor privado virtual (o VPS por sus siglas en inglés) no representa por sí misma una solución a los problemas relacionados al despliegue de la aplicación. Aunque el uso de contenedores en el servidor es una mejora significativa en términos de aprovechamiento de recursos y automatización de procesos, su implementación aún puede generar desafíos. En particular, no está bien definida una forma estandarizada de realizar el despliegue en el servidor utilizando tecnología de contenedores para equipos pequeños de desarrollo. Esto a menudo deriva a la implementación de malas prácticas, como ser:

- Clonar el código fuente del proyecto en el VPS, aun cuando ya deja de ser necesario.
- Realizar la construcción de la imagen del contenedor dentro del mismo servidor.

- No incluir en el contenedor parte de la aplicación, como ser el servidor web, el proxy reverso o la base de datos.
- Sobrecarga del servidor en memoria debido a la construcción de imágenes o procesos de codificación de los archivos de configuración del contenedor dentro del servidor.
- Sobrecarga del servidor en almacenamiento debido a que no se remueven los contenedores huérfanos o a que se clona el código fuente de todos los proyectos en el servidor.

Por lo recién mencionado, lo que se plantea en este trabajo es definir una forma eficiente tanto en tiempo como en recursos para realizar el despliegue de distintas aplicaciones en un VPS, construir el soporte tecnológico para llevarlo adelante y comprobarlo en un equipo de desarrollo software de la industria local.

1.2 Objetivo general

Desarrollar una propuesta de trabajo de despliegue para aplicaciones web que van a ser desplegadas en servidores Linux, mediante la utilización de tecnologías actuales de control de versiones y contenedores con el fin tanto de optimizar y simplificar el proceso de despliegue y la actualización de software en dichos entornos mejorando el rendimiento y disminuyendo la cantidad de errores.

1.3 Objetivos específicos

Para cumplimentar el objetivo general definido en el apartado anterior, se han propuesto la realización de determinados objetivos específicos, que lo componen y se indican a continuación.

1.3.1 (OE1)

Analizar las buenas prácticas de la industria del software que busca resolver las dificultades del despliegue de aplicaciones web y seleccionar las actividades que pueden ser implementadas en el marco de equipos pequeños de desarrollo de software.

1.3.2 (OE2)

Construir de forma iterativa e incremental un procedimiento para desplegar aplicaciones web con arquitectura monolítica en servidores privados virtuales. Operacionalizarlo a partir de un repositorio compartido con archivos de configuración base del contenedor y scripts de despliegue.

1.3.2 (OE3)

Establecer un enfoque de despliegue que se base en la utilización de un repositorio centralizado para almacenar todos los archivos relacionados con el proceso de despliegue de la aplicación, incluyendo su documentación.

1.3.3 (OE4)

Verificar y mejorar el procedimiento a partir de su uso en un equipo de trabajo de una empresa de la industria del software de la región.

2. Metodología

En este capítulo, se presenta la metodología de desarrollo utilizada para la construcción del proyecto. Se optó por el uso de un modelo de ciclo de vida evolutivo debido a su flexibilidad y adaptabilidad, especialmente en proyectos con alto grado de incertidumbre en cuanto a los requisitos finales del sistema. El modelo evolutivo permitió una entrega temprana del procedimiento, de la solución tecnológica o repositorio y una mayor flexibilidad en la incorporación de nuevas funcionalidades. En este capítulo se describen las etapas del proceso iterativo aplicado a través de la metodología, que incluyen la definición de los requisitos básicos del proyecto, el diseño de la estructura del repositorio, la evaluación y retroalimentación de los usuarios finales, la planificación y desarrollo de nuevas versiones, la realización de pruebas y validación y la entrega y documentación final del proyecto.

2.1 Ciclo de Vida Evolutivo

Para la construcción del proyecto se utilizó un modelo de ciclo de vida evolutivo, como se puede ver en la Fig. 2. Esta estrategia permite una mayor flexibilidad y adaptación a medida que se van agregando nuevas funcionalidades y características. Además, el modelo evolutivo proporciona un entorno propicio para el desarrollo de software incremental [8] y es ideal para proyectos que tienen un alto grado de incertidumbre en cuanto a los requisitos finales del sistema, ya que permite una entrega temprana de una versión parcialmente funcional del proyecto, que luego se va mejorando y evolucionando a medida que se reciben comentarios y retroalimentación de los usuarios finales.

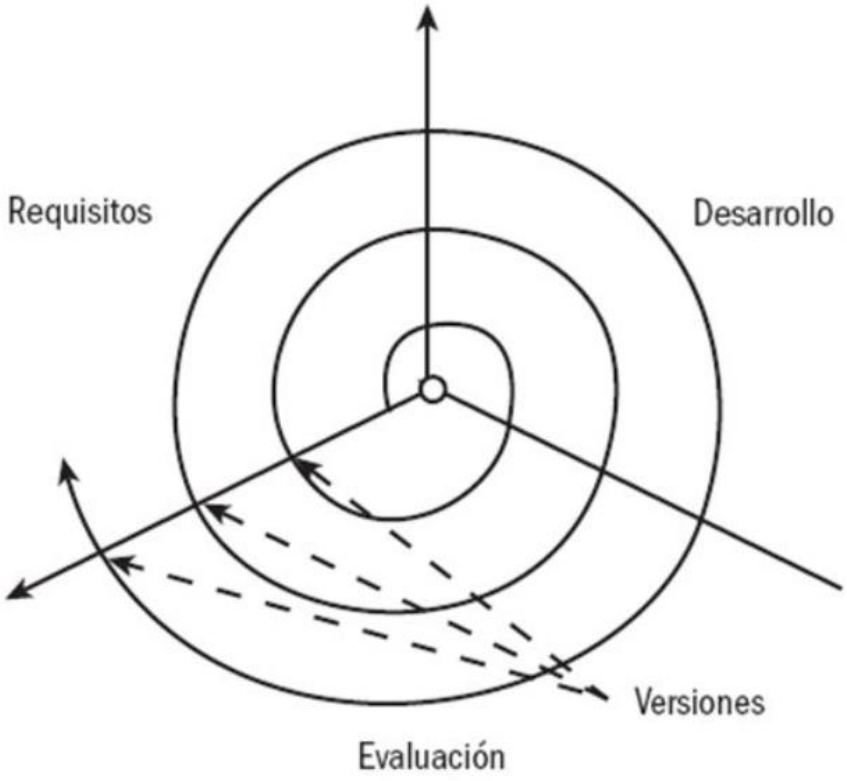


Fig. 2: Modelo de ciclo de vida evolutivo. Fuente: [8]

En el caso de este proyecto, el uso del modelo evolutivo permitió una entrega temprana del repositorio. Además, este enfoque permitió una mayor flexibilidad en cuanto a la incorporación de nuevas funcionalidades y características futuras.

Por lo recién mencionado se aplicó este modelo a través de un proceso iterativo que se detalla a continuación en la Tabla 1.

Tabla 1: Etapas de la metodología aplicada.

Etapa	Detalle
Desarrollo de una versión inicial	En esta etapa se definieron los requisitos básicos del proyecto y se diseñó la estructura de carpetas que tendría el repositorio. Se trabajó en el desarrollo de una versión parcialmente funcional del repositorio, que incluye las características y funcionalidades iniciales requeridas para el proyecto. Esta versión inicial servirá como base para las iteraciones y mejoras posteriores, en las que se agregarán nuevas funcionalidades y se mejorará la usabilidad y rendimiento del repositorio.
Evaluación y retroalimentación	En esta etapa, los usuarios finales, que en este caso fueron los miembros de los equipos de desarrollo de software, probaron la versión inicial del repositorio y proporcionan retroalimentación sobre su funcionamiento y características.
Planificación y desarrollo de nuevas versiones	En función de los comentarios y retroalimentación recibidos de los usuarios finales, se planificaron y desarrollaron nuevas versiones del repositorio que incluyen nuevas funcionalidades y mejoras.
Pruebas y validación	En cada iteración del proceso, se realizaron pruebas y validación para asegurarse de que el repositorio de despliegue cumpla con los requisitos y funcionalidades definidos para esa versión.
Entrega y documentación	Una vez que se cumplieron todos los requisitos y se agregaron todas las funcionalidades necesarias, se realiza la entrega final del proyecto.

2.1.1 Justificación de la elección del modelo evolutivo

El modelo evolutivo es una opción adecuada cuando el proyecto tiene un alto grado de incertidumbre y cambios constantes. En un proyecto de despliegue de aplicaciones web en servidores Linux, es común encontrarse con cambios en las configuraciones de los servidores, actualizaciones de software y otros ajustes que deben ser implementados. El modelo evolutivo permite hacer frente a estos cambios de manera más efectiva que otros modelos de ciclo de vida.

Además, el modelo evolutivo permite un enfoque más flexible y adaptable al desarrollo de software. En este modelo, los requisitos y la funcionalidad se van desarrollando de manera iterativa, permitiendo al equipo de desarrollo responder de manera más ágil a los cambios y requerimientos del proyecto. Esto se traduce en una mayor eficiencia y eficacia en el proceso de desarrollo de software.

Otra ventaja del modelo evolutivo es que se enfoca en la entrega continua y gradual del software, lo que permite un mayor grado de retroalimentación y validación temprana del producto. Esto significa que los errores o problemas en el desarrollo pueden ser detectados y resueltos de manera más oportuna, lo que resulta en un mejor producto final.

Finalmente, el modelo evolutivo también ofrece una mayor flexibilidad en cuanto a los recursos y tiempos necesarios para el desarrollo de software. Al permitir el desarrollo iterativo y modular, los recursos necesarios pueden ser asignados de manera más flexible, lo que se traduce en una mayor eficiencia y eficacia en el proceso de desarrollo.

3. Marco teórico

La automatización de procesos de despliegue es un aspecto crítico en el desarrollo de aplicaciones, especialmente cuando se trata de aplicaciones en contenedores. La automatización puede mejorar la velocidad y la fiabilidad del despliegue, y reducir el tiempo y los costos asociados con el desarrollo y el mantenimiento de aplicaciones. En este capítulo se abordarán diferentes metodologías y tecnologías utilizadas en la automatización de procesos de despliegue, incluyendo los contenedores, plataformas para despliegue de aplicaciones web, infraestructura como código y la integración y despliegue continuo. Se presentarán también trabajos relacionados con el tema. Además, se enseña un resumen de los conocimientos disciplinarios específicos adquiridos a lo largo de las distintas asignaturas cursadas y aplicados en el desarrollo de este trabajo final de especialización.
Este capítulo busca proporcionar una comprensión más profunda del marco teórico detrás de la automatización de procesos de despliegue y sus aplicaciones en el mundo del desarrollo de software.

3.1 Conocimiento disciplinar específico

- Procesos de desarrollo de software: Se tomaron ideas promovidas por las metodologías ágiles de desarrollo de software en cuanto a tratar de crear un ambiente multidisciplinario entre los miembros del equipo. Se fomenta que todos los desarrolladores puedan desplegar el proyecto en Internet de manera sencilla, ya sea para entornos de desarrollo, pruebas o producción

- Prueba de software: La importancia de escribir un código fuente de calidad y documentar los procesos realizados ha sido una lección clave que se ha aplicado en este trabajo. Además de la automatización de estos procesos para agilizar el trabajo en general.

- Aplicaciones de Cloud Computing: Los conocimientos adquiridos en esta materia, específicamente en el área de redes y servidores, han sido de gran importancia y han permitido abordar de manera efectiva los aspectos técnicos y tecnológicos de la investigación.

- Gestión del conocimiento: Se buscó Generar un flujo de conocimiento dentro de la organización de explícito a explícito, o sea compartir y transferir conocimientos de

manera efectiva entre los miembros del equipo. Esto de acuerdo al modelo de gestión de conocimiento denominado SECI [9] (por sus siglas en inglés Socialization, Externalization, Combination, and Internalization).

- Desarrollo avanzado de Aplicaciones: Los conocimientos adquiridos en esta materia fueron fundamentales para abordar la arquitectura y estructuración del proyecto. Se utilizaron los conceptos aprendidos sobre la creación de diagramas de proyectos de software, lo que permitió una planificación más eficiente y una comprensión clara de la estructura del proyecto.

- Prueba de Software: La importancia de escribir un código fuente de calidad y documentar los procesos realizados ha sido una lección clave que se ha aplicado en este trabajo. Además de la automatización de estos procesos para agilizar el trabajo en general. Se realizaron diversas pruebas para asegurar el funcionamiento del proyecto en diversos ambientes.

3.2 Metodologías y tecnologías estudiadas

3.2.1 Contenedores

Las tecnologías de contenedores ofrecen una agilidad sin precedentes en el desarrollo y la ejecución de aplicaciones. Los mismos nos son especialmente útil a la hora de desarrollar y desplegar nuestro sistema porque la implementación de aplicaciones web requiere un entorno de ejecución que incluya solamente algunas dependencias específicas para cada aplicación, como ser servidores web, soporte del lenguaje de programación, bases de datos y otros componentes; y no necesariamente todo un sistema operativo. Mediante el uso de herramientas de contenedores como ser Docker [10], LXC [11] o Podman [12], podemos empaquetar una aplicación web y su entorno de ejecución en “imágenes” o paquetes similares a pequeñas máquinas virtuales que puede ser desplegadas rápidamente en cualquier entorno de trabajo [7].

La llegada de los contendores al mundo del desarrollo, cambió la forma en la que se despliegan y mantienen las aplicaciones web modernas, a continuación, podemos mencionar algunas de las implementaciones más importantes actualmente.

3.2.1.1 LXC

Pioneros en esta tecnología, los contenedores de Linux (Linux Container o LXC) permiten empaquetar y aislar las aplicaciones junto con todo el entorno de tiempo de ejecución, es

decir, con todos los archivos que requieren para ejecutarse. Esto permite mover la aplicación que se encuentra dentro del contenedor entre los entornos (de desarrollo, de prueba, de producción, etc.), sin perder ninguna de sus funciones. Mediante la virtualización, se crea una instancia que garantiza la portabilidad y coherencia de cada contenedor en todo momento [13].

3.2.1.3 Docker

Docker emplea el concepto de "imágenes" para generar contenedores a partir de un proyecto de software. Se pueden crear múltiples contenedores utilizando una misma imagen. Estas imágenes se almacenan e indexan en un repositorio en línea, como DockerHub [14], GitLab Container Registry [15] o incluso en un registro personalizado en un servidor privado.
Posteriormente, desde un servidor, es posible descargar estas imágenes desde sus respectivos registros y ejecutarlas en dicho servidor. El proceso descrito anteriormente se representa en la Fig. 3.

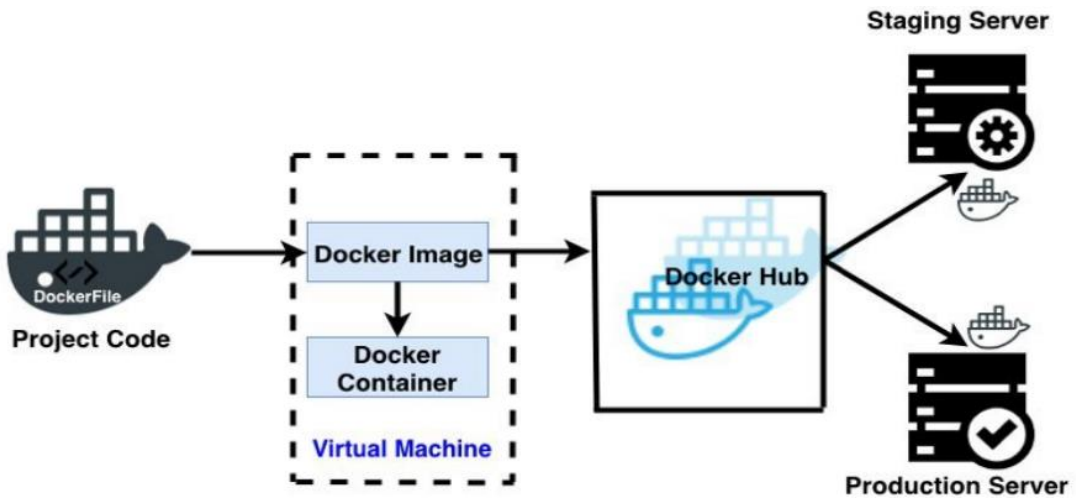


Fig. 3: Ciclo de vida en Docker. Fuente [16]

3.2.1.2 Podman

Podman es un motor de contenedores que se ejecuta sin un proceso demonio (*daemonless*) para desarrollar, administrar y ejecutar contenedores en sistemas Linux. Es de fuente abierta y desarrollado por Red Hat.
Para la creación de este motor de contenedores, se han descentralizado todos los componentes necesarios para la gestión de dichos contenedores y los han individualizado en componentes más pequeños que se utilizarán solo cuando sean necesarios. Esta descentralización nos ofrece un gran número de ventajas como ser un servicio más liviano aún que otras soluciones de contenedores, o como se mencionó anteriormente, que Red Hat

ha decidido desarrollar su herramienta sin depender de un servicio. Esto permite desligarse de la marca Docker (tenemos un archivo Containerfile.yml en lugar de Dockerfile.yml), permitiéndonos crear contenedores agnósticos [12].

Aunque Podman ofrece ventajas significativas en comparación con Docker, este último sigue siendo el estándar predominante para el manejo de contenedores en la actualidad. Esto se debe en gran parte a que Docker es ampliamente utilizado por la comunidad de desarrolladores y por los principales proveedores de computación en la nube, tales como Amazon web Services (AWS), Microsoft Azure y Google Cloud Platform (GCP).

3.2.2 Plataformas para despliegue de aplicaciones web

Como se ha visto, cuando se trata de desplegar aplicaciones web en Internet, hay diversas opciones disponibles. Para conocer las preferencias de los equipos de desarrollo en la región, se realizó una breve encuesta a ocho equipos de tres empresas. Los resultados obtenidos sobre el tipo de plataforma más utilizado para esta tarea se muestran en la Fig. 4.

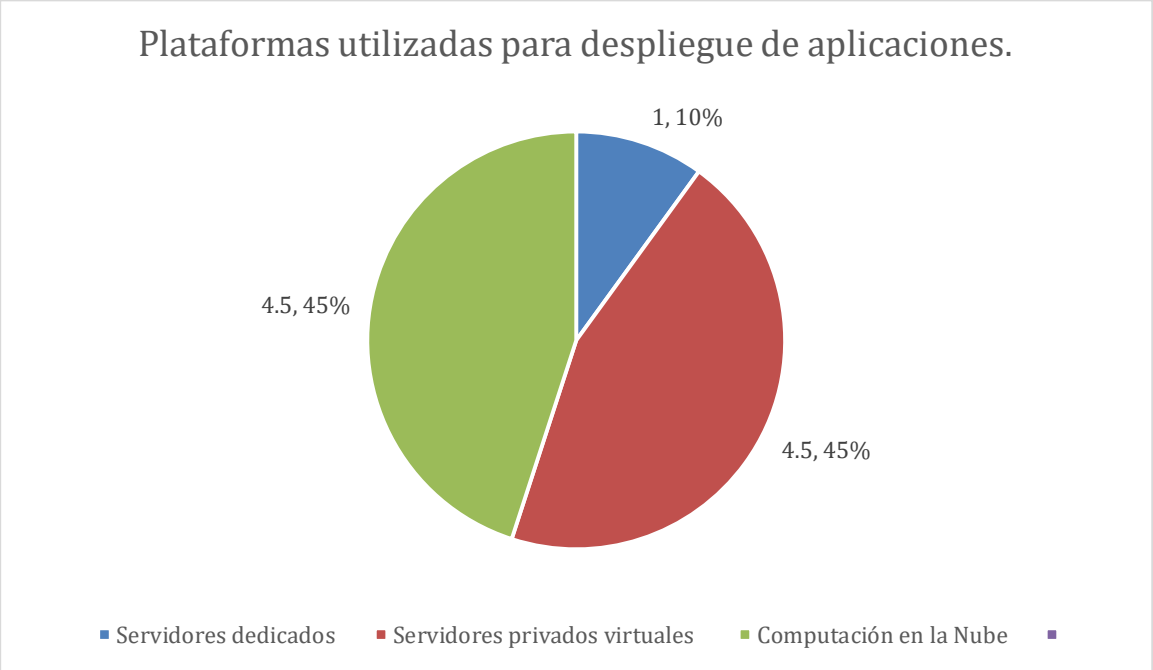


Fig. 4: Plataformas utilizadas para despliegue de aplicaciones web.

3.2.2.1 Servidores dedicados

Un servidor dedicado es una máquina física que se utiliza exclusivamente para alojar una aplicación o un sitio web. A diferencia de otras opciones de alojamiento, como el alojamiento compartido o los servidores virtuales, un servidor dedicado no se comparte con otras aplicaciones o sitios web. Esto significa que el servidor dedicado ofrece un mayor nivel de control y personalización, ya que el propietario del servidor tiene acceso completo a todos los

recursos y configuraciones del servidor. Además, un servidor dedicado también puede ofrecer un mayor nivel de seguridad y privacidad. Sin embargo, un servidor dedicado también puede ser más costoso que otras opciones de alojamiento debido al costo de la infraestructura física y su mantenimiento.

La elección de servidores dedicados para el despliegue de aplicaciones no es muy frecuente, siendo su uso principal cuando se requiere alojar aplicaciones críticas que manejan información sensible o confidencial.

3.2.2.2 Servidores privados virtuales (VPS)

Un servidor privado virtual es una partición virtual dentro de un servidor físico, que actúa como si fuera un servidor dedicado con su propio sistema operativo, espacio en disco, memoria y ancho de banda. Cada VPS puede ser configurado y personalizado de manera individual, lo que permite a los usuarios tener más control sobre su alojamiento web.

Los mismos ofrecen un equilibrio entre la funcionalidad y el costo de un servidor dedicado y el alojamiento compartido. Los usuarios tienen la libertad de instalar su propio software y personalizar la configuración del servidor, mientras comparten los recursos físicos del servidor con otros usuarios.

Entre los principales usos de los servidores privados virtuales en el mundo del desarrollo web se incluye tanto desarrollo y pruebas de aplicaciones como también Implementación y despliegue de aplicaciones.

3.2.2.3 Computación en la Nube

La computación en la nube (también conocida como "Cloud Computing" en inglés) es un modelo de computación que proporciona servicios de tecnología de la información a través de Internet.

Los servicios de computación en la nube se ofrecen a través de servidores remotos gestionados por proveedores de servicios en la nube, como Amazon Web Services, Microsoft Azure, Google Cloud, entre otros. Estos proveedores ofrecen una variedad de servicios, como almacenamiento de datos, procesamiento de datos, alojamiento de sitios web y aplicaciones, análisis de datos, inteligencia artificial, aprendizaje automático y mucho más.

Los principales modelos de computación en la nube se ven representados en la Tabla 2.

Tabla 2: modelos de computación en la nube.

Modelo de Computación en la Nube	Descripción	Ejemplo
IaaS (Infraestructura como servicio)	Este modelo proporciona acceso a recursos informáticos, como servidores, almacenamiento y redes, a través de Internet. El usuario es responsable de la configuración y administración de su infraestructura en la nube.	Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform
PaaS (Plataforma como servicio)	Este modelo proporciona una plataforma de desarrollo en la nube que permite a los desarrolladores crear y ejecutar aplicaciones sin tener que preocuparse por la infraestructura subyacente. El usuario es responsable de la gestión de la aplicación.	Heroku, Google App Engine, Microsoft Azure App Service
SaaS (Software como servicio)	Este modelo proporciona aplicaciones de software completas que se ejecutan en la nube y están disponibles para su uso a través de Internet. El usuario no tiene que preocuparse por la gestión de la infraestructura o la aplicación.	Google Workspace (anteriormente G Suite), Microsoft 365, Salesforce

Entre las principales ventajas de la computación en la nube se incluyen la escalabilidad, la flexibilidad, la eficiencia y el ahorro de costos. Los usuarios pueden escalar sus recursos de acuerdo a sus necesidades, pagar solo por los recursos que utilizan y acceder a servicios de alta calidad, como bien se explica en [17]:

> *La computación en nube ofrece nuevas oportunidades para desplegar aplicaciones escalables de forma eficiente, permitiendo que las aplicaciones empresariales puedan ajustar dinámicamente sus recursos de cómputo bajo demanda.*

Sin embargo, a pesar de las ventajas que ofrece la computación en la nube, es importante tener en cuenta algunas de sus posibles desventajas en comparación con los servicios de VPS, como ser:

- Muchas veces es difícil estimar el costo final de los servicios utilizados. Y dependiendo de qué servicio se utilice es posible que se presenten costos adicionales imprevistos.
- La dependencia de un proveedor de servicios en la nube puede dificultar la transición a otro proveedor en caso de ser necesario. Además, algunos proveedores pueden ofrecer servicios propietarios que no son interoperables con otros servicios, lo que limita la capacidad del usuario para utilizar herramientas y aplicaciones de terceros.
- La curva de aprendizaje que conlleva la utilización de estos servicios, configuración y administración de la infraestructura en la nube, requiere habilidades y conocimientos técnicos específicos, lo que puede resultar en una inversión de tiempo y recursos significativa.

3.2.3 Infraestructura como código (IaC)

La infraestructura como código es una práctica que implica la gestión y el aprovisionamiento automatizados de la infraestructura de TI utilizando código y herramientas de desarrollo de software, según explica Red Hat en [18],

> *La infraestructura como código permite a los equipos de TI gestionar la infraestructura de manera más eficiente y escalable, ya que el código se puede versionar, probar y desplegar automáticamente. Además, esta práctica también ayuda a reducir errores y a aumentar la velocidad y la agilidad en la entrega de aplicaciones y servicios de TI.*

Las herramientas de infraestructura como código, como Docker y Docker Compose, desempeñan un papel crucial en el desarrollo y la orquestación de software nativo y a escala [19]. Con ellas podemos automatizar la creación y administración de infraestructura de la aplicación a través de código, en lugar de configurar manualmente cada recurso, como servidores, redes y almacenamiento. Esto permite que la infraestructura se trate como cualquier otro componente de software, lo que significa que puede ser probada, versionada, dividida en diferentes ambientes y actualizada de manera más fácil y rápida.

3.2.4 Integración y Despliegue continuo (CI/CD)

Como parte de la transformación Agile en los últimos años, hemos visto a las organizaciones de TI adoptar principios de integración continua en su ciclo de vida de entrega de software, lo que ha mejorado la eficiencia de los equipos de desarrollo [20].

Las prácticas continuas, es decir, la integración, entrega e implementación continuas, son las prácticas de la industria de desarrollo de software que permiten a las organizaciones lanzar nuevas funciones y productos de manera frecuente y confiable. Con el creciente interés y la literatura sobre prácticas continuas, es importante revisar y sintetizar sistemáticamente los enfoques, herramientas, desafíos y prácticas informados para adoptar e implementar prácticas continuas [21].

Actualmente, existen diversas herramientas para la implementación de CI/CD, tales como Jenkins, GitHub Actions y GitLab CI/CD. No obstante, es factible llevar a cabo una implementación de despliegue continuo a través de scripts, lo que permite acelerar el proceso y evitar la necesidad de configurar una herramienta específica para ello. En este enfoque, los scripts se encargan de realizar tareas como dar de baja los contenedores, descargar las imágenes más actuales y volver a levantarlos, lo que permite una implementación de despliegue continuo efectiva y eficiente para aplicaciones web desplegadas en servidores VPS.

3.3 Trabajos relacionados

3.3.1 CapRover

CapRover es una plataforma de PaaS de código abierto que se utiliza para simplificar la implementación y el alojamiento de aplicaciones web en un servidor o clúster de servidores. CapRover proporciona una interfaz de usuario web fácil de usar para administrar la plataforma y las aplicaciones, y ofrece características como la automatización del ciclo de vida de las aplicaciones, el equilibrio de carga, el escalado automático y la gestión de bases de datos. CapRover también admite la implementación de aplicaciones en contenedores Docker y proporciona integraciones con herramientas de CI / CD para una implementación continua [22].

La instalación de este software en el servidor resulta interesante, sin embargo, presenta algunas limitaciones importantes. Por ejemplo, no soporta todos los lenguajes y tecnologías comúnmente utilizados en empresas de desarrollo. Además, se requiere de una gran capacidad en el servidor para su ejecución y a medida que se despliegan más aplicaciones en esta solución, más memoria se ocupará, lo que limita el servidor para un solo tipo de despliegue, inhabilitándolo para la utilización de otras tareas, como la instalación de una base de datos para su uso distribuido. Por lo tanto, la decisión de utilizar este software debe hacerse en un servidor específico para este propósito.

3.3.2 Engage

Muchas aplicaciones modernas se crean mediante la combinación de paquetes y servicios desarrollados de forma independiente que se distribuyen en muchas máquinas con interdependencias complejas. El ensamblaje, la instalación y la administración de tales aplicaciones son difíciles y, por lo general, se realizan manualmente o escribiendo scripts personalizados. Engage es un sistema para configurar, instalar y administrar pilas de aplicaciones complejas. Engage consta de tres componentes: un modelo específico de dominio para describir los metadatos de los componentes y las dependencias entre componentes; un algoritmo basado en restricciones que toma una especificación de instalación parcial y calcula un plan de instalación completo; y un sistema de tiempo de ejecución que coordina la implementación de la aplicación en varias máquinas y administra el sistema implementado. Mediante el modelado explícito de los metadatos de configuración y las dependencias entre componentes, Engage permite la verificación estática de las configuraciones de la aplicación y la generación automatizada, basada en restricciones, de planes de instalación en varias máquinas. Esto reduce el tedioso proceso manual de configuración, instalación y administración de aplicaciones [23].

3.3.3 CTCD-e MLOps Pipeline

En relación a este trabajo de especialización, pero enfocándose en el ámbito del despliegue de modelos de aprendizaje automático, se llevó a cabo un estudio en la Universidad de Helsinki, en Finlandia, en el año 2023. Este estudio propone una canalización de MLOps (Machine Learning Operations) independiente de la nube y de código abierto que permite a los usuarios volver a entrenar y volver a implementar sus modelos de ML de manera flexible. Se aplicó la metodología Design Science, que consiste en identificar el problema, definir los objetivos de la solución, implementar, demostrar y evaluar la solución. La solución resultante es un *pipeline* MLOps llamada CTCD-e MLOps Pipeline. Puede adaptar de forma autónoma los modelos de aprendizaje automático a los datos de producción dinámicos iniciando automáticamente el reentrenamiento de los modelos de aprendizaje automático cuando su rendimiento se degrada. También puede aplicar pruebas A/B automáticamente para el rendimiento de los modelos reentrenados en producción y los implementa por completo solo cuando superan a sus predecesores. Además, CTCD-e MLOps Pipeline permite a sus usuarios configurar de manera flexible el reentrenamiento y la reimplementación del modelo, así como la prueba A/B de producción de los modelos reentrenados en función de varios requisitos [24].

Esta herramienta se implementa en un entorno de computación en la nube utilizando Kubernetes. Sin embargo, nuestra propuesta se centra en la automatización de despliegues en un solo servidor, lo cual difiere de este enfoque.

3.4 Conclusiones

En la revisión de los trabajos relacionados tanto como en las investigaciones sobre las principales tecnologías involucradas en este documento, se encontró que la mayoría de las soluciones de despliegue de aplicaciones se basan en la tecnología de Computación en la Nube. Sin embargo, también se concluye que, pese a la popularidad actual de la Computación en la Nube y las arquitecturas de Microservicios, todavía resulta conveniente para muchos equipos y proyectos web la utilización de arquitecturas monolíticas y el despliegue de los mismos en servidores privados virtuales. En este sentido, la solución propuesta en este trabajo se enfoca en el despliegue de aplicaciones en VPS, ofreciendo una alternativa más asequible y personalizable para aquellos que buscan una solución de despliegue de aplicaciones que se adapte a sus necesidades específicas de manera rápida y efectiva. Al utilizar nuestro repositorio con archivos base de contenedores de Docker, configuraciones de proxys, y scripts automatizados, los equipos de desarrollo pueden desplegar fácilmente su aplicación en un entorno de VPS, lo que les permite tener más control sobre la configuración y el rendimiento de su aplicación, como también sobre la migración hacia distintos ambientes y servidores

.

4. Desarrollo y Resultados

En este capítulo se presenta la implementación de la propuesta planteada, incluyendo aspectos técnicos relevantes, iteraciones del proceso de desarrollo, enlace al repositorio del proyecto y ejemplos de utilización. Además, se muestran los resultados obtenidos en la ejecución de la propuesta, incluyendo la evaluación del desempeño y la efectividad de la solución implementada.

4.1 Propuesta de implementación

Como se ha mencionado anteriormente, la mayoría de las aplicaciones desarrolladas por los equipos se despliegan en servidores VPS, lo cual puede generar confusión debido a la variedad de formas en que se puede llevar a cabo esta tarea, especialmente cuando se utilizan múltiples aplicaciones en un mismo servidor. Si bien es cierto que se puede mejorar este proceso mediante el uso de contenedores, la simple utilización de Docker no garantiza una solución óptima y sistemática para el despliegue de una o varias aplicaciones en un servidor.

El objetivo principal de este trabajo final de especialización es, como se indicó en el capítulo 1, estandarizar la forma de desplegar aplicaciones web monolíticas en servidores VPS para un equipo pequeño de desarrollo software. Para lograr esto, se propone seguir un método similar al utilizado por equipos de desarrollo de aplicaciones web distribuidas y de gran tamaño. Estos equipos emplean una variedad de recursos, tales como microservicios, múltiples servicios de computación en la nube, herramientas de orquestación de contenedores como Kubernetes, múltiples instancias de la aplicación y bases de datos distribuidas, entre otros [17].

En lugar de enfocarnos en tecnologías específicas, la presente propuesta consiste en imitar el proceso definido que estos equipos han incorporado a su flujo de trabajo. De esta manera, se busca estandarizar el despliegue de aplicaciones web monolíticas en servidores VPS de manera sistemática y eficiente. En lugar de centrarnos exclusivamente en herramientas o tecnologías concretas, el objetivo es seguir un enfoque metodológico que permita alcanzar esta meta. Siguiendo este enfoque, se espera mejorar significativamente la eficiencia y la escalabilidad del despliegue de aplicaciones web monolíticas en servidores VPS.

4.1.1 Enfoque metodológico adoptado

En proyectos y equipos de gran envergadura, además de utilizar repositorios para alojar el código fuente de las distintas partes de la aplicación, como el *frontend*, los servicios de *backend* y las interfaces de programación de aplicaciones (API por sus siglas en inglés), entre otros, cuentan con repositorio específico que almacena todas las configuraciones necesarias

para el despliegue de la aplicación. Este enfoque permite una gestión más eficiente de los procesos de desarrollo y despliegue, lo que resulta en un proceso más ordenado y efectivo. En general, estas configuraciones y archivos incluyen manifiestos del orquestador de contenedores como Kubernetes [25], manifiestos de administradores de paquetes como Helm [26], configuraciones de proxys y balanceadores de carga, configuraciones de malla de servicios o Service Mesh como Linkerd [27] o Istio [28], archivos de infraestructura como código como Cloudformation [29] o Terraform [30], configuraciones de plataforma de intercambio de mensajes entre microservicios o Brocker como Apache Kafka [31] o RabbitMQ [32], entre otros.

4.1.2 Adaptación de la metodología aplicada al contexto de un equipo de desarrollo pequeño

La idea principal consiste en utilizar el método del "repositorio de despliegue" en una escala reducida, mediante la creación de un "repositorio esqueleto" de uso general para múltiples proyectos. Este repositorio actuará como una estructura básica o esqueleto para el repositorio de despliegue específico de cada proyecto.

El repositorio cumple la función de plantilla y punto de partida, al incluir un conjunto mínimo de archivos, directorios y configuraciones necesarios para el despliegue del proyecto. Su propósito es brindar una base sólida que pueda ser personalizada y expandida para satisfacer las necesidades específicas de cada proyecto en particular.

En este caso, se incluyen principalmente archivos Dockerfile y docker-compose.yml con configuraciones genéricas que pueden ser personalizadas mediante un archivo “.env” de variables de entorno, adaptándose a las necesidades del proyecto. Además, se proporcionan archivos de configuración para el servidor web y proxy reverso Nginx, junto con un contenedor encargado de agregar y renovar la seguridad en la capa de transporte (SSL/HTTPS) a través de Certbot [33] de la autoridad certificadora Let’s Encrypt [34]. También se ofrece otro proxy reverso Nginx para gestionar la redirección de solicitudes, como se puede apreciar en la Fig. 5, junto con scripts de Bash que permiten a los desarrolladores automatizar tareas repetitivas en lugar de tener que ejecutar comandos manualmente. Estos scripts ejecutan tareas, tales como:

- Ejecutar el proceso de construcción de las imágenes necesarias para la aplicación.
- Subir las imágenes construidas al registro de contenedores (o Docker Registry) especificado para su posterior despliegue.
- Ejecutar los contenedores con la versión más reciente de la aplicación para garantizar su correcto funcionamiento.

- Realizar copias de seguridad de la base de datos para asegurar la integridad de los datos en caso de algún incidente.

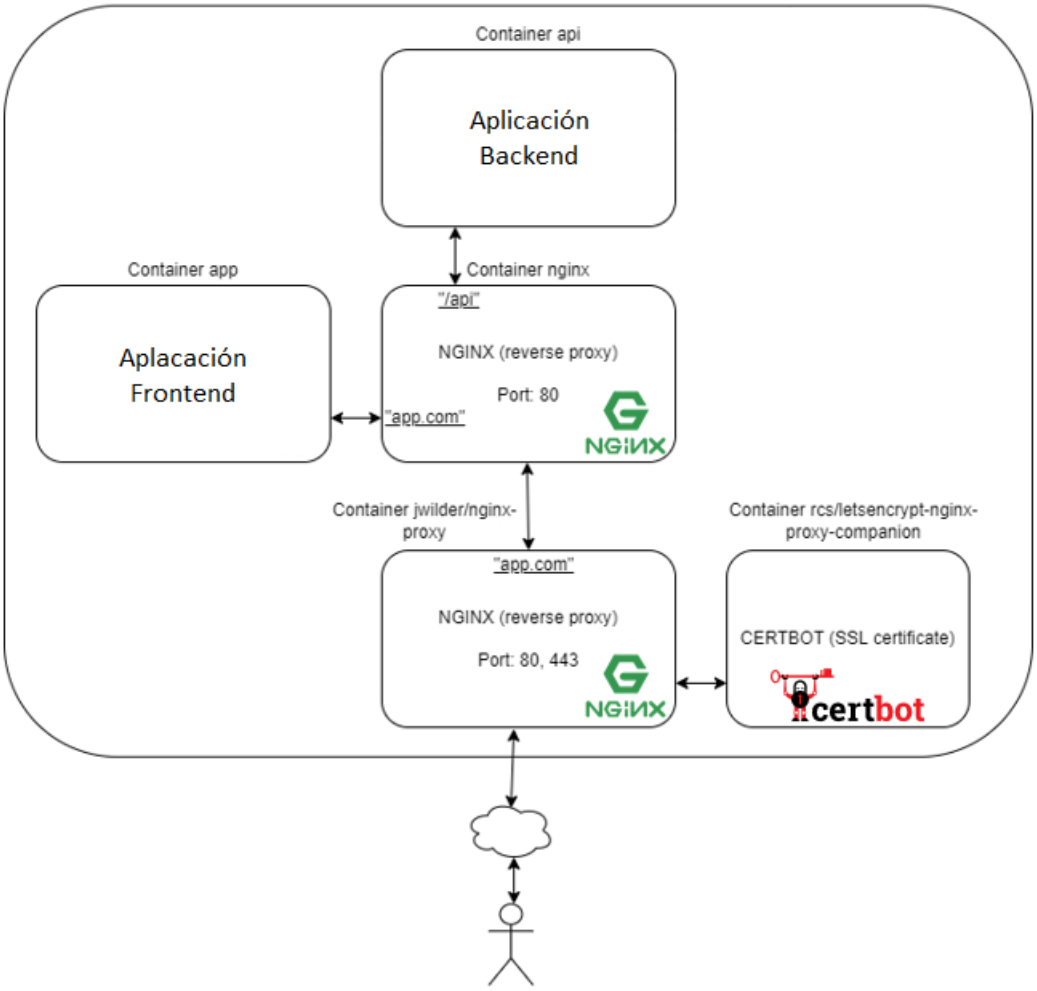


Fig. 5: Estructura de contenedores de servidor y redireccionamiento.

Otra de las encuestas realizadas a equipos de desarrollo de software de la región tuvo como objetivo conocer la estructura que utilizan en sus proyectos. De acuerdo con los resultados obtenidos, el 70% de los proyectos sigue una arquitectura monolítica, el 20% utiliza una arquitectura *serverless*, y el 5% emplean arquitectura de tipo microservicios. Estos resultados se pueden observar en la Fig. 6: Arquitecturas de proyectos más utilizadas.

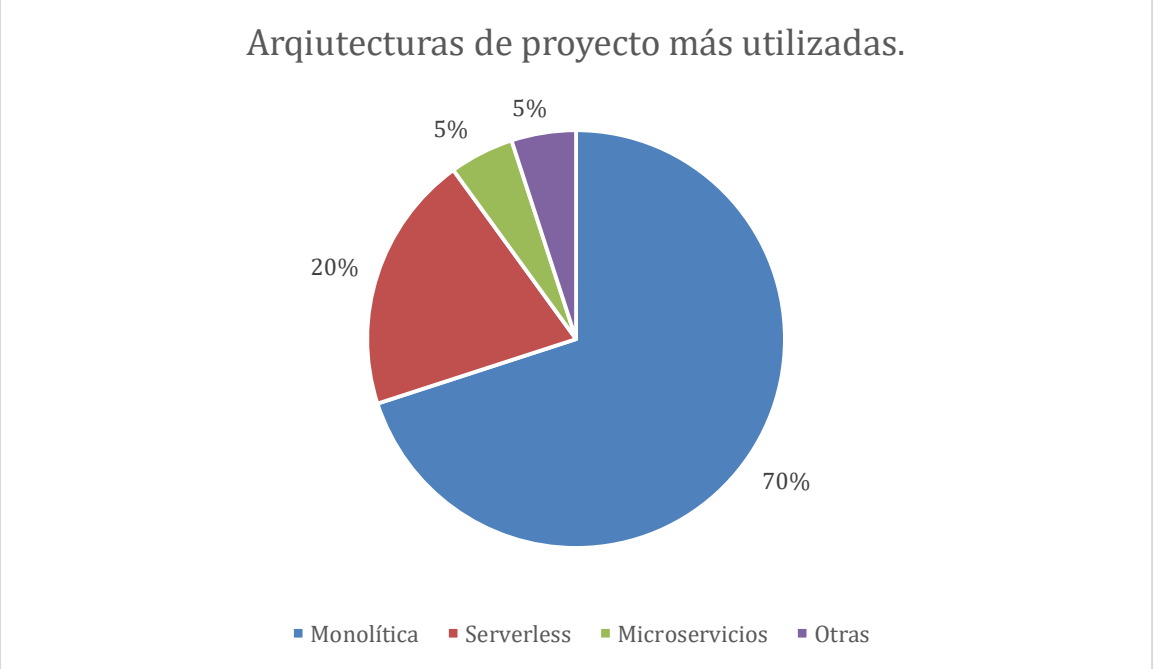


Fig. 6: Arquitecturas de proyectos más utilizadas

También se consultó sobre cómo se organizaban las partes de los proyectos monolíticos, ya sea utilizando un único repositorio o repositorios separados para la aplicación cliente o *frontend* y para la aplicación servidor o *backend*, independientemente de si se utiliza una arquitectura de Rest API o GraphQL. Los resultados se muestran en la Fig. 7, indican que actualmente el 30% de los proyectos monolíticos se dividen en varios repositorios (multi-repositorio), uno para cada parte del proyecto, mientras que un 70% utiliza un único repositorio (mono-repositorio) para todo el proyecto.

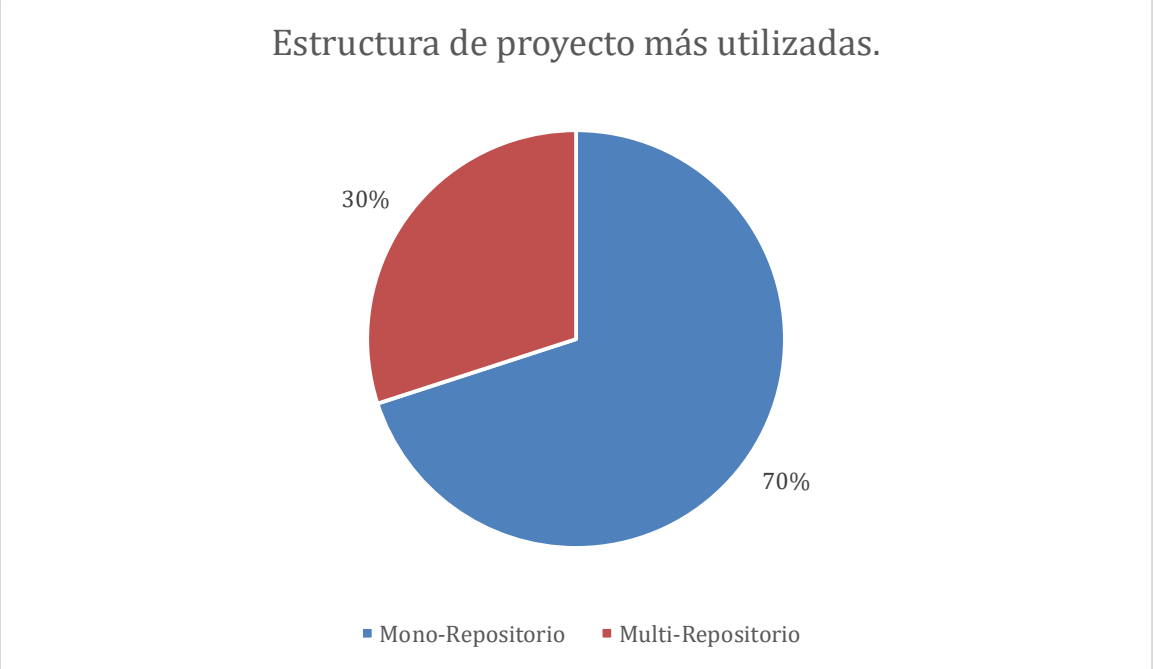


Fig. 7: Estructura de proyectos.

El proceso de desarrollo de software puede variar significativamente según el contexto, especialmente en lo que respecta a la estructura del equipo y las prácticas que se siguen. Sin embargo, en proyectos monolíticos de múltiples repositorios, es común y ampliamente utilizado el siguiente esquema de organización de carpetas, como se muestra en la Fig. 8.

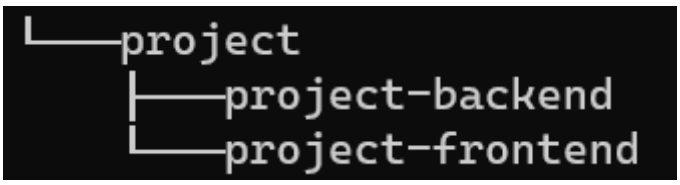


Fig. 8: Esquema de carpetas en proyectos monolíticos.

La estructura monolítica mencionada previamente ofrece la ventaja de separar el proyecto en dos partes: el servidor (backend) y el cliente (frontend). Según Martin Fowler en [9], esta división en líneas de trabajo separadas es fundamental para el flujo de trabajo de los equipos de desarrollo de software. Es común que los desarrolladores se especialicen en una sola parte del proyecto, ya sea el backend o el frontend. Por lo tanto, el enfoque de múltiples repositorios resulta beneficioso para el equipo de desarrollo y permite aplicar buenas prácticas de organización de proyectos.

Siguiendo esta idea, hemos añadido el repositorio de despliegue (*deployment*) propuesto en este trabajo al mismo nivel que los repositorios de *backend* y *frontend* ya conocidos, como se muestra en la Fig. 9.

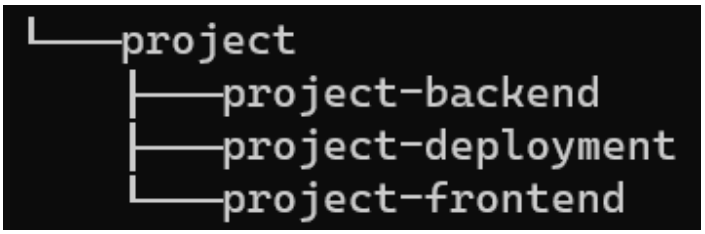


Fig. 9: Estructura de carpetas propuesta.

El repositorio de despliegue será el único que se clone en el servidor donde se alojará la aplicación web. Este repositorio se utilizará para gestionar el despliegue de la aplicación, tanto para la primera vez como para sus futuras actualizaciones, además de posibles tareas y procedimientos programados, como copias de seguridad de datos, entre otras.

4.1.3 Registro de contenedores

La elección del registro de contenedores es un aspecto crucial del proyecto, ya que de él dependerá la subida y almacenamiento de las imágenes de Docker de cada proyecto que utilice el repositorio base de despliegue. Si bien Docker Hub es el registro más conocido y popular, se decidió utilizar Gitlab como registro de contenedores debido a su gratuidad y a la posibilidad de establecer el repositorio como privado. Se construyó la Tabla 3 para comparar ambas opciones.

Tabla 3: Comparación entre DokerHub y Gitlab Container Registry.

Característica	Docker Hub	GitLab Container Registry
Costo	Gratis hasta cierto límite, luego se debe pagar.	Gratuito para repositorios privados y públicos.
Soporte de imagen	Soporta imágenes Docker oficiales y de la comunidad.	Soporta imágenes Docker y Helm Chart.
Tiempo de espera de construcción de imagen	No hay garantía de tiempo de espera.	Construcción rápida de imágenes debido al uso de los recursos propios.
Integración con CI/CD	Soporta integraciones con herramientas populares de CI/CD, como Jenkins y Travis CI.	Integrado con GitLab CI/CD para una experiencia de implementación completa en la misma plataforma.
Seguridad	Escaneo de seguridad y validación de imágenes en busca de vulnerabilidades.	Escaneo de seguridad y validación de imágenes en busca de vulnerabilidades, junto con la opción de habilitar la autenticación de dos factores.
Características adicionales	Repositorios públicos y privados disponibles.	Soporte para la creación de grupos, proyectos y subgrupos.
Comunidad y soporte	Gran comunidad y documentación disponible.	Gran comunidad y soporte disponible de GitLab.

4.1.4 Mejoras en el enfoque de trabajo

La Fig. 10 ilustra el proceso de despliegue y la metodología de trabajo mencionados recientemente. Aunque a simple vista se aprecia un diagrama más complejo que el esquema de la *Fig. 1*, el mismo ofrece varias ventajas en comparación con el anterior, tales como:

- No es necesario subir el código fuente del proyecto al servidor.
- Solo se clona el repositorio de despliegue en el servidor, ahorrando espacio y tiempo.

- No es necesario instalar programas y dependencias adicionales en el servidor, lo que facilita la gestión del mismo.
- El despliegue se realiza en segundos gracias a la automatización del proceso.
- Se puede migrar de servidor fácilmente, ya que todo lo necesario para el despliegue se encuentra en el repositorio de despliegue.

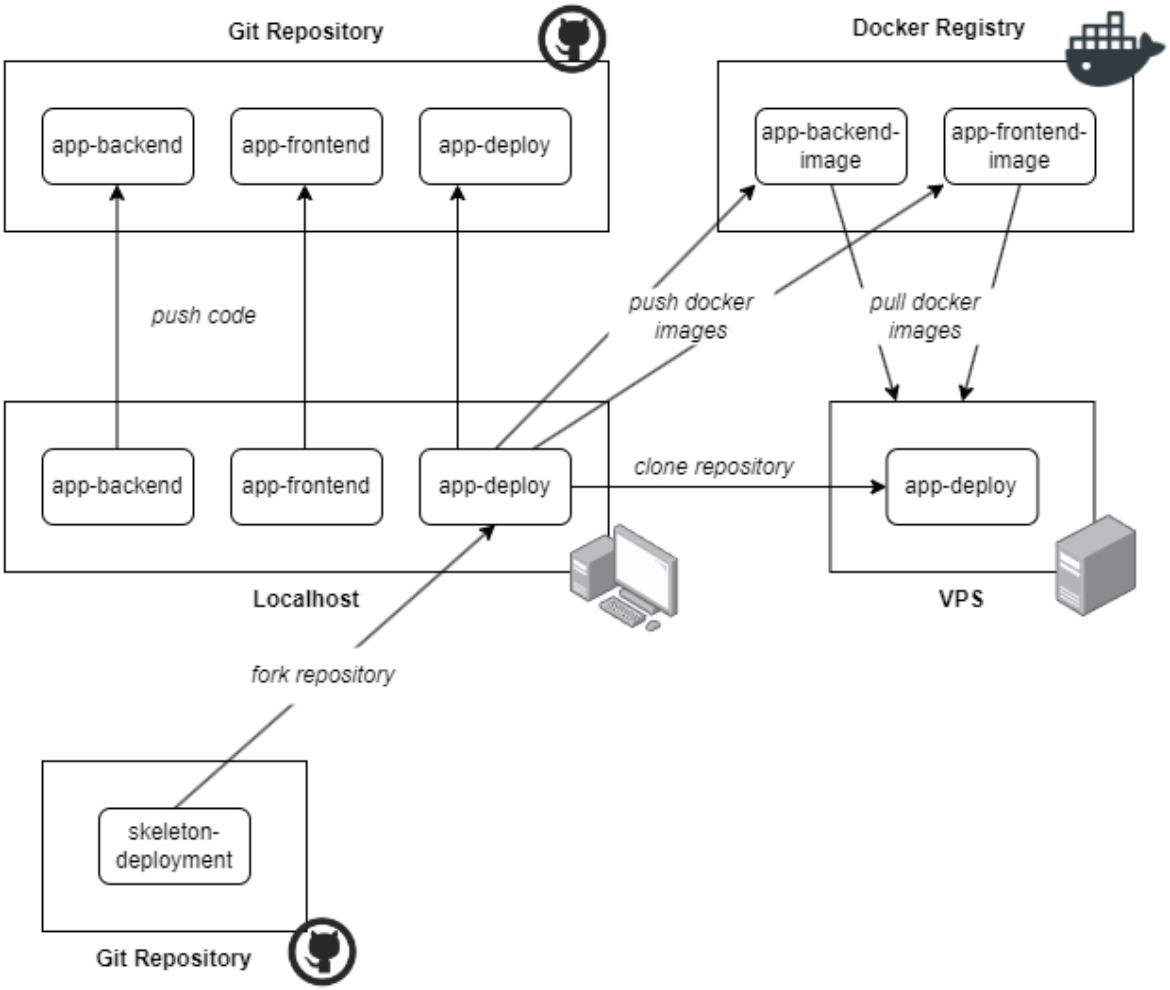


Fig. 10: Diagrama de metodología propuesta para el despliegue de aplicaciones web monolíticas a VPS.

4.1.5 Aspectos relevantes del desarrollo del proyecto

Durante el desarrollo de la solución, se utilizó un lenguaje de scripting que funcionaba principalmente en Linux, aunque también debía ser compatible con Windows y Mac, para garantizar la compatibilidad en todos los sistemas operativos. Se encontró que Bash funcionaba en Mac, pero no en Windows con PowerShell. Sin embargo, se pudo solucionar este problema mediante la utilización en Windows de la consola de Git Bash, la cual es ampliamente utilizada hoy en día por los desarrolladores independientemente del sistema operativo que utilicen.

4.2 Desarrollo en Iteraciones

Se presenta la descripción detallada del desarrollo del proyecto utilizando un enfoque iterativo basado en el modelo de ciclo de vida evolutivo presentado en capítulos anteriores. El desarrollo del proyecto se realizó en distintas iteraciones, cada una enfocada en agregar nuevas funcionalidades y mejoras al repositorio.

4.2.1 Iteración Nro. 1: Diseño inicial

Para iniciar el proyecto, se creó una carpeta y se recopilaron diversos scripts de bash, Dockerfiles y docker-compose.yml utilizados previamente en proyectos anteriores. Esto se consideró como una primera aproximación al trabajo, ya que inicialmente se pretendía simplemente organizar los archivos en un repositorio para utilizarlos de forma independiente. Sin embargo, surgió la idea de crear un repositorio base de despliegue, lo que llevó a la evolución del proyecto.

En la primera etapa, se procedió a ordenar y reescribir algunos scripts y archivos de Docker para su uso general. Sin embargo, estos archivos estaban preconfigurados con información específica del proyecto original donde fueron creados, lo que significaba que, si se utilizaban de esta manera, los desarrolladores tendrían que actualizar y configurar muchos archivos manualmente, lo que sería una tarea tediosa. Para evitar esto, se decidió crear una carpeta de configuración que contenía un archivo "*.env*", donde se definirían todas las variables de entorno del proyecto. Al hacer esto, se minimizó el esfuerzo ya que los desarrolladores solo tendrían que editar un solo archivo para utilizar los scripts y archivos de Docker en cualquier proyecto sin tener que preocuparse por configurar cada archivo uno por uno.

En la Fig. 11 se puede apreciar tanto el archivo utilizado para la configuración del proyecto, como la estructura del mismo en sus primeras instancias.

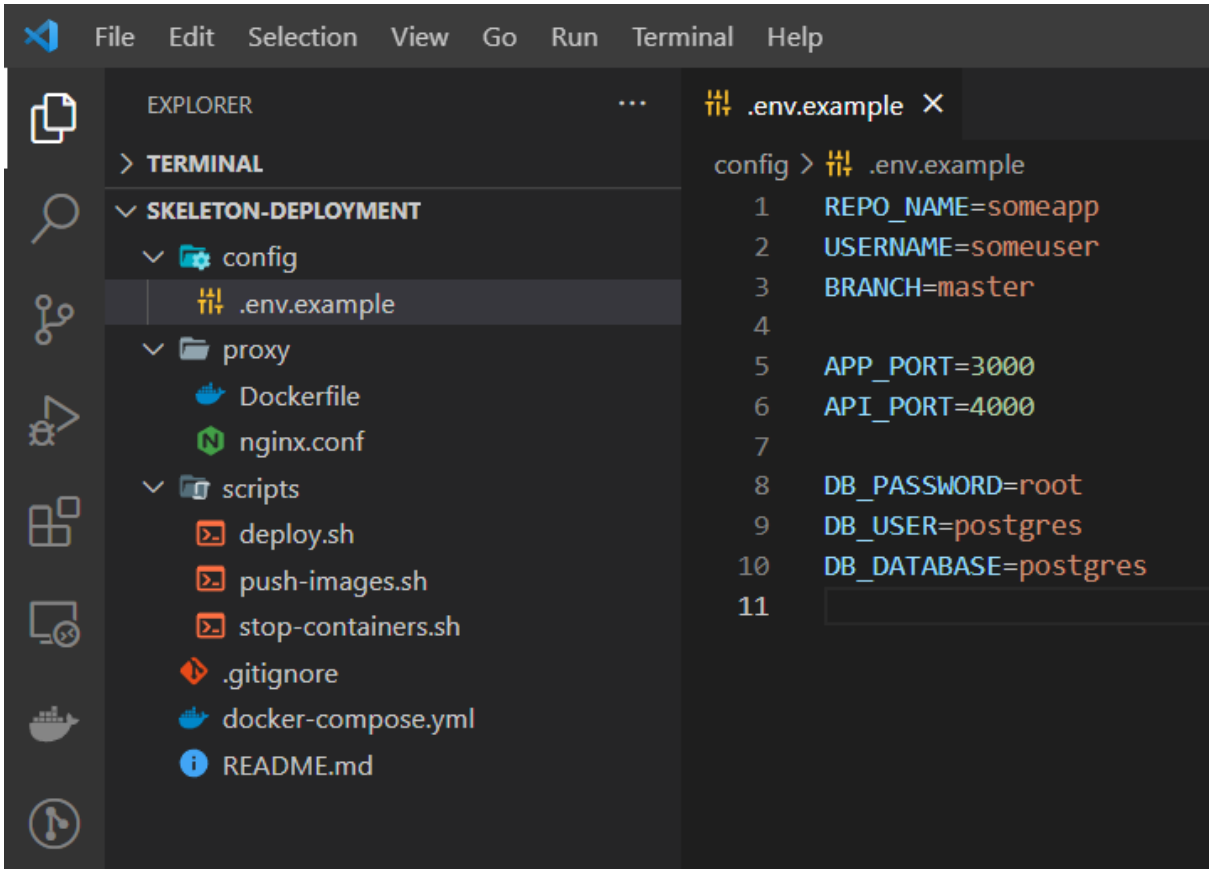


Fig. 11: Archivo de configuración y estructura principal.

4.2.1.1 Detalles de la estructura del proyecto en su primera instancia

En esta primera instancia, las carpetas y archivos que contenía el proyecto eran los siguientes:

- **/config**: carpeta que contiene el archivo de configuración principal del proyecto.
    - **.env.example**: archivo de configuración de variables de entorno. Se utiliza este nombre con la idea de reemplazar el mismo por uno de nombre ".env" que contenga los valores reales de cada variable de entorno a utilizarse.
- **/proxy**: carpeta donde se alojan los archivos relacionados al proxy que se utilizara para conectar y redireccionar las solicitudes a la aplicación cliente y a la aplicación servidor según corresponda.
    - **Dockerfile**: archivo con imagen del servidor web / proxy reverso, Nginx.
    - **nginx.conf**: archivo de configuración del proxy reverso.
- **/scripts**: carpeta que contiene scripts útiles para el proceso de despliegue.
    - **deploy.sh**: usado para desplegar la aplicación.
    - **push-images.sh**: usado para subir las imágenes del proyecto al registro de contenedores.
    - **stop-containers.sh**: usado para detener contenedores en ejecución.

- **.gitignore**: archivo utilizado para indicar a Git qué archivos y carpetas del proyecto deben ser ignorados.
- **Docker-compose.yml**: archivo que contiene toda la estructura de contenedores del proyecto.
- **README.md**: documentación del proyecto.

4.2.2 Iteración Nro. 2: Ambientes de despliegue

La primera versión del proyecto resultó útil para el despliegue de la aplicación, ya que contenía los scripts necesarios para facilitar todo el proceso. Sin embargo, tenía la limitación de que solo permitía la implementación en un solo entorno.

Después de encuestar a los equipos de desarrollo de la región, así como basarse en el conocimiento propio y en expertos en el tema, se llegó a la conclusión de que es común establecer tres ambientes principales para el despliegue: Desarrollo, Escena y Producción (o Development, Staging y Production, respectivamente). En [24] se explica el propósito de cada uno de estos entornos, y se ve ilustrado en Fig. 12.

- Development: Ambiente donde se realiza el desarrollo y prueba del software antes de su lanzamiento. Aquí se llevan a cabo las pruebas unitarias y de integración, y se pueden hacer cambios y correcciones en el código.
- Staging: Ambiente donde se realizan pruebas más completas del software antes de su lanzamiento. En este ambiente se llevan a cabo pruebas de aceptación del usuario final y se pueden hacer ajustes finales antes de pasar a producción.
- Production: Ambiente de producción, donde el software se utiliza de forma activa por los usuarios finales. En este ambiente se espera que el software funcione sin problemas y que cualquier problema que surja sea atendido de manera inmediata.

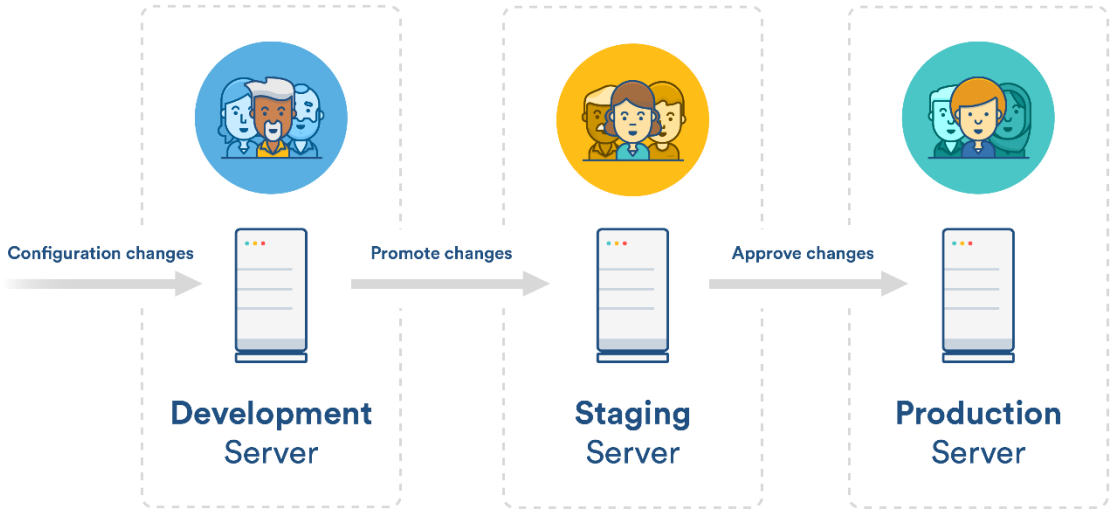


Fig. 12: Entornos de despliegue [35].

Por la razón mencionada anteriormente, se procedió a separar el archivo docker-compose.yml en tres entornos distintos. La estructura resultante del proyecto se refleja en la Fig. 13.

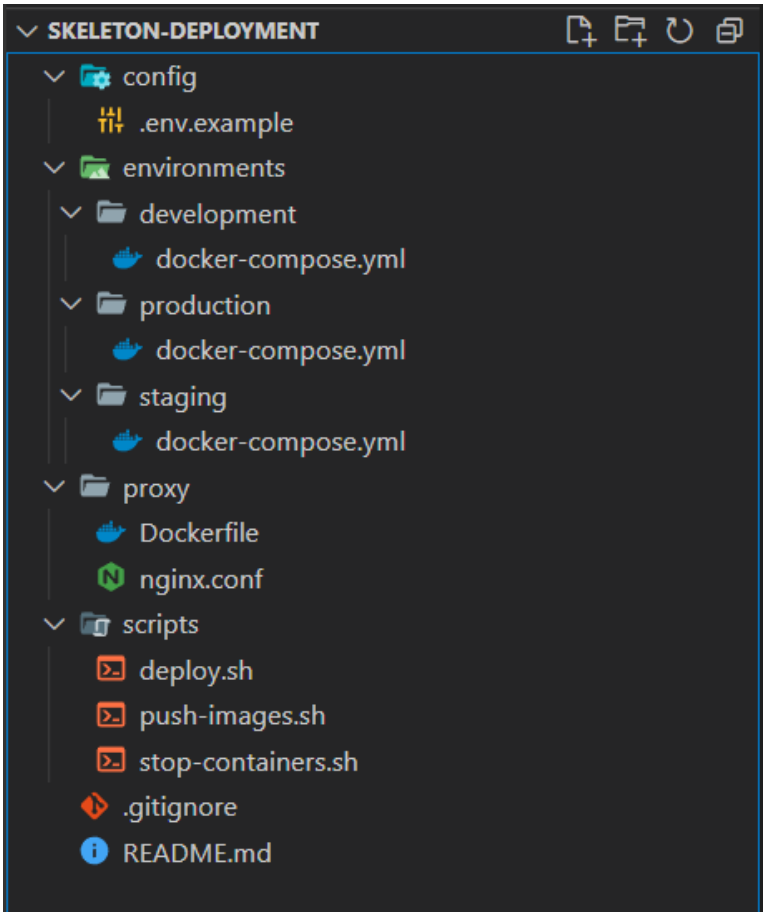


Fig. 13: Estructura del proyecto luego de agregar los ambientes.

4.2.2.1 Detalles de la estructura del proyecto agregado los ambientes

- **/config**: carpeta que contiene el archivo de configuración principal del proyecto.
    - **.env.example**: archivo de configuración de variables de entorno. Se utiliza este nombre con la idea de reemplazar el mismo por uno de nombre ".env" que contenga los valores reales de cada variable de entorno a utilizarse.
- **/environments**: carpeta que contiene los ambientes de despliegue disponibles.
    - **/development/docker-compose.yml**: archivo con estructura de contenedores para ambiente de desarrollo.
    - **/staging/docker-compose.yml**: archivo con estructura de contenedores para ambiente de escena.
    - **/production/docker-compose.yml**: archivo con estructura de contenedores para ambiente de producción.
- **/proxy**: carpeta donde se alojan los archivos relacionados al proxy que se utilizara para conectar y redireccionar las solicitudes a la aplicación cliente y a la aplicación servidor según corresponda.
    - **Dockerfile**: archivo con imagen del servidor web / proxy reverso, Nginx.

    - **nginx.conf**: archivo de configuración del proxy reverso.
- **/scripts**: carpeta que contiene scripts útiles para el proceso de despliegue.
    - **deploy.sh**: usado para desplegar la aplicación.
    - **push-images.sh**: usado para subir las imágenes del proyecto al registro de contenedores.
    - **stop-containers.sh**: usado para detener contenedores en ejecución.
- **.gitignore**: archivo utilizado para indicar a Git qué archivos y carpetas del proyecto deben ser ignorados.
- **Docker-compose.yml**: archivo que contiene toda la estructura de contenedores del proyecto.
- **README.md**: documentación del proyecto.

4.2.3 Iteración Nro. 3: Carpeta con utilidades

En esta iteración se consideró la posibilidad de incluir archivos de utilidades que son ampliamente utilizados en la mayoría de los proyectos web. Estas utilidades podrían ser tareas de ejecución programada, como backups, o archivos de ejemplo que se utilizan

comúnmente en la mayoría de los proyectos web. La Fig. 14: Estructura del proyecto luego de agregar las utilidades. muestra la estructura resultante.

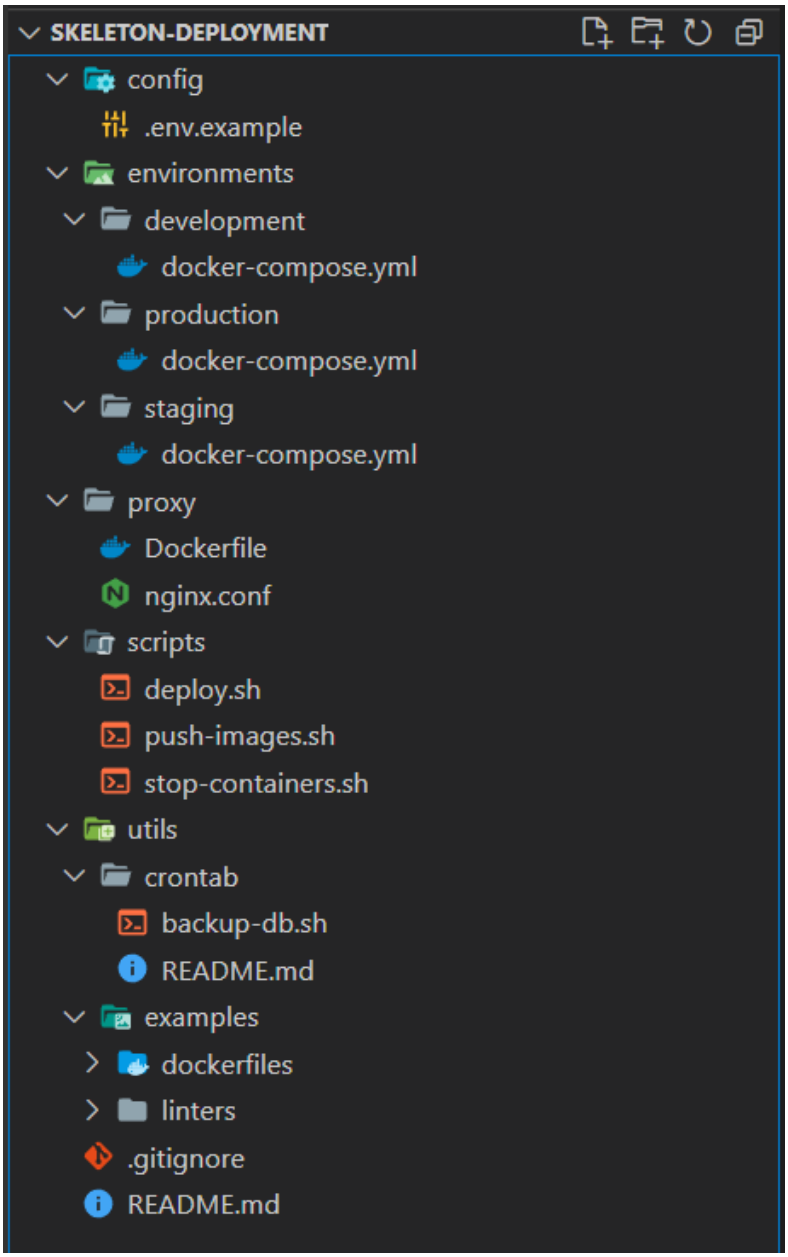


Fig. 14: Estructura del proyecto luego de agregar las utilidades.

4.2.3.1 Detalles de la estructura del proyecto agregado los ambientes

- **/config**: carpeta que contiene el archivo de configuración principal del proyecto.
    - **.env.example**: archivo de configuración de variables de entorno. Se utiliza este nombre con la idea de reemplazar el mismo por uno de nombre ".env" que contenga los valores reales de cada variable de entorno a utilizarse.
- **/environments**: carpeta que contiene los ambientes de despliegue disponibles.

    - **/development/docker-compose.yml**: archivo con estructura de contenedores para ambiente de desarrollo.
    - **/staging/docker-compose.yml**: archivo con estructura de contenedores para ambiente de escena.
    - **/production/docker-compose.yml**: archivo con estructura de contenedores para ambiente de producción.
- **/proxy**: carpeta donde se alojan los archivos relacionados al proxy que se utilizara para conectar y redireccionar las solicitudes a la aplicación cliente y a la aplicación servidor según corresponda.
    - **Dockerfile**: archivo con imagen del servidor web / proxy reverso, Nginx.
    - **nginx.conf**: archivo de configuración del proxy reverso.
- **/scripts**: carpeta que contiene scripts útiles para el proceso de despliegue.
    - **deploy.sh**: usado para desplegar la aplicación.
    - **push-images.sh**: usado para subir las imágenes del proyecto al registro de contenedores.
    - **stop-containers.sh**: usado para detener contenedores en ejecución.
- **/utils**: carpeta que contiene utilidades comúnmente usadas en proyectos web.
    - **/crontab**: carpeta con scripts para tareas de ejecución programada.
        - **backup-db.sh**: script para realizar un backup de la base de datos.
        - **README.md**: guía de utilización y configuración del script.
    - **/examples**: carpeta con scripts para tareas de ejecución programada.
        - **/dockerfiles**: ejemplos de archivos Dockerfile.
        - **/linters**: ejemplo de archivos de análisis estático de código fuente.
- **.gitignore**: archivo utilizado para indicar a Git qué archivos y carpetas del proyecto deben ser ignorados.
- **Docker-compose.yml**: archivo que contiene toda la estructura de contenedores del proyecto.
- **README.md**: documentación del proyecto.

4.2.4 Iteración Nro. 4: Prueba y ajustes finales

Durante la cuarta iteración del proyecto, se enfocó en la organización y seguridad del mismo, así como en la mejora de su rendimiento técnico. En primer lugar, se decidió agregar "/pgdata" a la lista de archivos y carpetas ignorados en el archivo .gitignore. Esta carpeta almacena los datos de la base de datos que se encuentra dentro del contenedor, y su inclusión en la lista de ignorados ayuda a prevenir la exposición de información sensible.

Además, se añadió el archivo ".env" al listado de ignorados, lo que permite que cada proyecto pueda configurar sus propias variables de entorno de manera individualizada. Esta medida aumenta la seguridad de los datos almacenados en el proyecto y permite una mayor flexibilidad para los desarrolladores.

En cuanto a la parte técnica del proyecto, se realizaron revisiones y pruebas de los scripts y archivos docker-compose. Durante este proceso, se detectaron pequeños errores que fueron corregidos de manera inmediata. También se llevaron a cabo refactorizaciones para mejorar la calidad del código.

Un listado de los errores y soluciones aplicadas se puede encontrar en la Tabla 4.

Tabla 4: Listado de errores más importantes.

Error	Solución	Elemento modificado
Puertos codificados de manera estática en archivos	Se parametrizaron los puertos en un archivo de configuración	docker-compose.yml, .env
Nombres de servicios codificados de manera estática en archivos	Se parametrizaron los nombres de servicios en los archivos docker-compose.yml y nginx.conf	docker-compose.yml, nginx.conf
Problemas de configuración en el proxy Nginx	Se revisó y corrigió la configuración del servidor proxy Nginx para el enrutamiento correcto	nginx.conf, Dockerfile

El listado de refactorizaciones aplicadas y de las ventajas obtenidas se puede apreciar en la Tabla 5.

Tabla 5: Listado de refactorizaciones más importantes.

Refactorización	Ventaja obtenida
División de script en múltiples comandos	Mayor modularidad y facilidad de mantenimiento
Condicional para comandos según el sistema	Mayor compatibilidad con sistemas Windows y Unix
Eliminación de codificación estática mediante archivo .env	Mayor flexibilidad y configuración centralizada
Uso de funciones reutilizables en Bash	Reducción de duplicación de código y mayor modularidad
Implementación de volúmenes en contenedores	Persistencia de datos y mayor eficiencia en el almacenamiento

4.2.5 Iteración Nro. 5: Documentación y lanzamiento

En la etapa final del proyecto, se realizaron los ajustes finales antes de subirlo a Internet. Se evaluaron diferentes opciones de alojamiento de repositorios de control de versiones, como Github, Gitlab y Bitbucket. A pesar de que el proyecto utiliza Gitlab para almacenar las imágenes de Docker debido a sus registros de imágenes gratuitos, se eligió Github como la plataforma de alojamiento para el repositorio base de despliegue debido a su popularidad en la comunidad de proyectos de código abierto. Además, antes de subir el proyecto, se revisó y reescribió la documentación para proporcionar a los usuarios del repositorio toda la información necesaria para incorporar el proyecto a sus propios proyectos y agilizar el proceso de despliegue.

A continuación, se presenta una transcripción de la documentación, junto con una captura de pantalla que se muestra en la Fig. 15 para una mejor visualización.

4.2.5.1 Documentación del proyecto

````
# Skeleton-Deployment

Este repositorio contiene las configuraciones base necesarias para
desplegar aplicaciones web monolíticas en una VPC.


## Entornos Configurados

Actualmente, el repositorio soporta tres entornos:

- `desarrollo`

- `preproducción ` (o `qa`)

- `producción`


## Cómo usar el repositorio

Siga estos pasos para usar el repositorio en su proyecto:


### 1) Agregue el Repositorio a su Proyecto

Puede bifurcar el repositorio y agregarlo a su proyecto, o clonarlo y
luego eliminar la carpeta `.git`.

Suponiendo que su proyecto tiene la siguiente estructura:

```

/proyecto/

    - proyecto-back

    - proyecto-front

```

Debe agregar el repositorio de la siguiente manera:

```

/proyecto/

    - proyecto-back

    - proyecto-front

    - proyecto-deploy

```
````

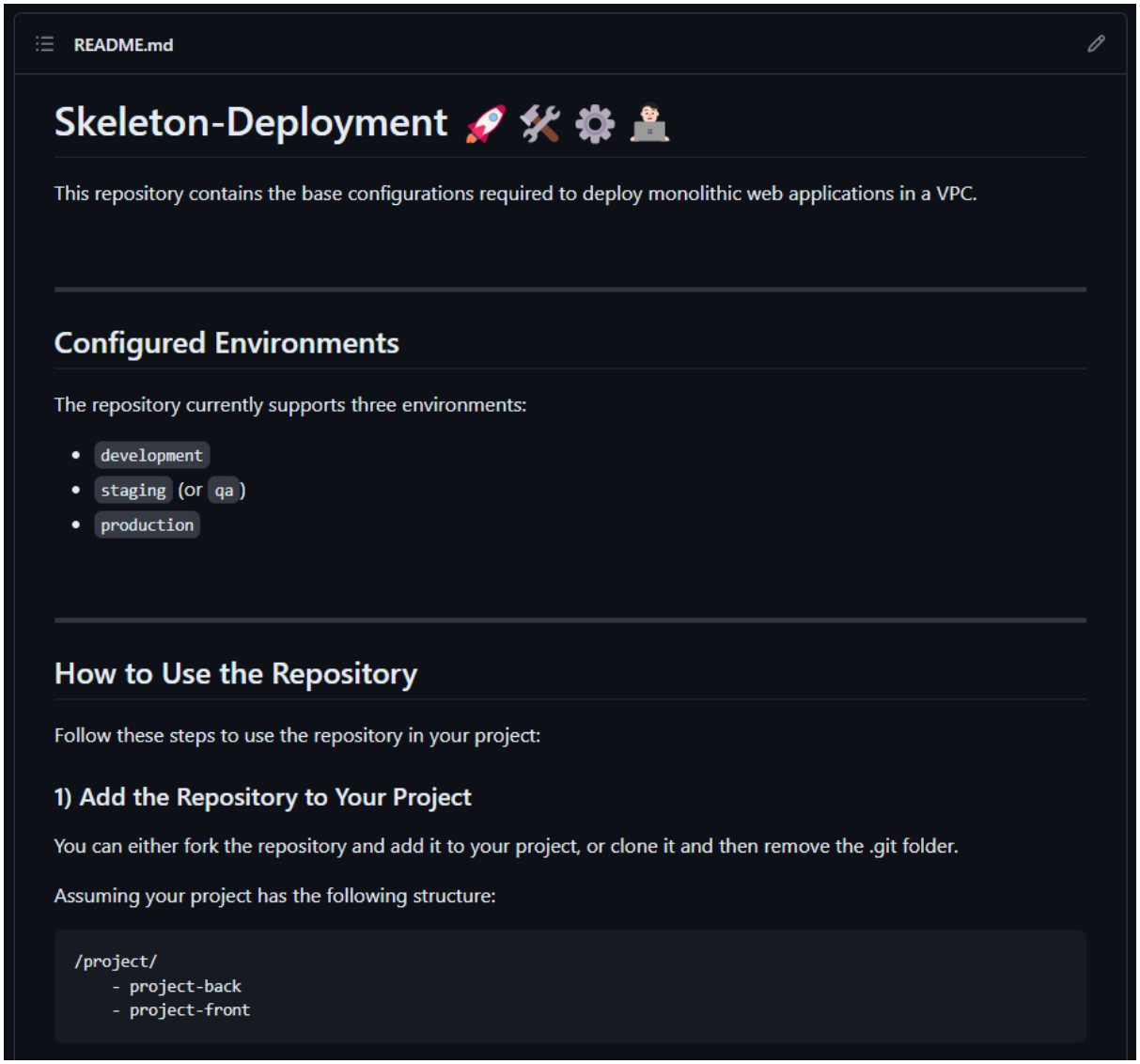


Fig. 15: Documentación del proyecto.

Una vez que se completaron las últimas refactorizaciones, tanto en la documentación como en la estructura de las carpetas, los scripts y otros archivos del proyecto, se subió la primera versión al repositorio. Esta versión está lista para ser clonada y utilizada, como se muestra en la Fig. 16.

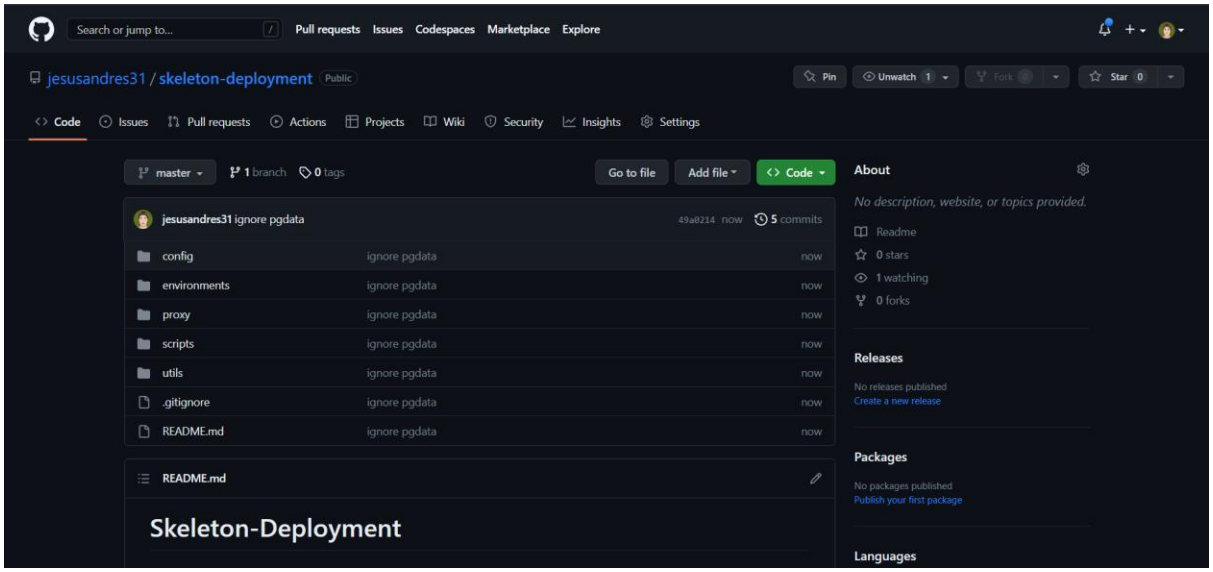


Fig. 16: Repositorio subido a GitHub.

4.2.5.2 Enlace al repositorio del proyecto

A continuación, compartimos el enlace al repositorio en línea donde se encuentra alojado el código fuente del proyecto. Este repositorio ha sido creado con el objetivo de permitir que cualquier persona interesada en la solución pueda acceder al código fuente y tener una visión detallada del trabajo que se ha llevado a cabo. Además, a modo de fomentar la colaboración y el intercambio de conocimientos, se invita a cualquiera que lo desee a aportar cambios o mejoras al proyecto a través del repositorio compartido:

https://github.com/jesusandres31/skeleton-deployment

Para acceder al código fuente del proyecto, simplemente puede clonar el repositorio desde GitHub en el directorio de su equipo que prefiera. Para hacerlo, solo necesita ingresar el siguiente comando en la terminal:

```
git clone https://github.com/jesusandres31/skeleton-deployment.git
```

De esta manera, podrá descargar una copia local del repositorio en su equipo y explorar el código fuente del proyecto.

4.3 Ejemplo de utilización

En esta sección se presentará la demostración del uso del repositorio de despliegue a través de su aplicación en dos proyectos, los cuales serán desplegados en un servidor privado virtual de la plataforma Google Cloud Platform (GCP), propiedad de la empresa Devlights de

Corrientes, Argentina. Además, se llevará a cabo el despliegue de uno de estos proyectos en un servidor VPS de Donweb, empresa de servicios de alojamiento de páginas web de Argentina. De esta manera, se podrá mostrar en la práctica la utilidad y eficacia de este repositorio en un entorno real de despliegue de aplicaciones. En resumen, se realizarán tres despliegues en total de dos proyectos.

El primer proyecto, llamado LiberApp, es una aplicación que está siendo desarrollada para la gestión de clientes de un estudio contable de la región. Se desplegará inicialmente en un entorno de desarrollo y pruebas en el servidor de GCP, y posteriormente se desplegará en el servidor de Donweb para su uso en ambiente de producción.

El segundo proyecto se llama TimeTracker y se utiliza para el registro de horas de trabajo de los empleados de Devlights. En este caso, se llevará a cabo una actualización del despliegue. Actualmente, esta aplicación ya se encuentra desplegada de forma clásica en ambiente de producción de GCP. Lo que se hará ahora es utilizar el repositorio de despliegue para realizar una nueva implementación en el mismo servidor. De esta forma, se podrá demostrar cómo el repositorio facilita y agiliza el proceso de despliegue en un ambiente real de producción.

La Tabla 6 proporciona un resumen completo de los tres despliegues realizados con el repositorio, incluyendo el nombre de la aplicación, el ambiente y el servidor en el que se llevó a cabo cada uno de ellos.

Tabla 6: Despliegues realizados.

Nombre de la aplicación web.	**Ambiente de despliegue.**	**Servidor utilizado.**
TimeTracker	Production	Google Cloud Platform
Liber App	Development	Google Cloud Platform
Liber App	Staging	Donweb

4.3.1 TimeTracker (despliegue en ambiente de producción)

TimeTracker es una herramienta empleada por Devlights para el registro de horas y tareas de sus empleados. Se trata de una aplicación web monolítica que consta de dos partes: una aplicación cliente desarrollada con React.js, y una aplicación servidor construida con Node.js.

Para la base de datos se optó por MongoDB, un tipo de base de datos NoSQL. La estructura del proyecto se ilustra en la Fig. 17.

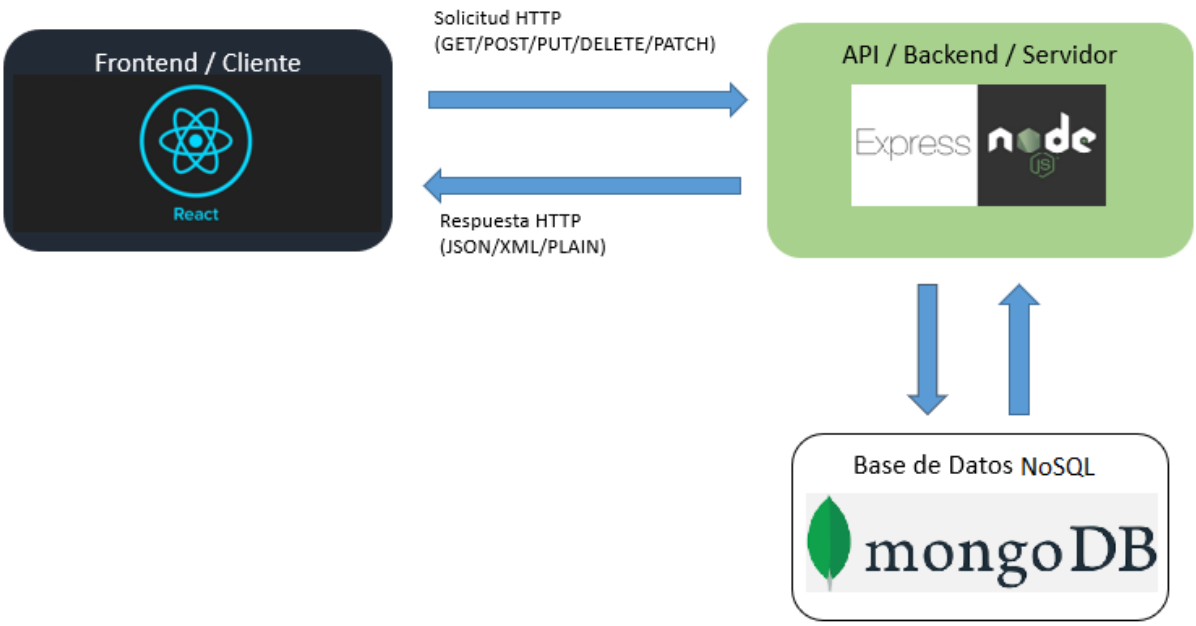


Fig. 17: Estructura de TimeTracker.

La aplicación empleaba una solución de despliegue similar a la propuesta en este trabajo, a través de un script de bash que llevaba a cabo el despliegue directamente en el servidor VPC de Google Cloud Platform. Para ello, se instalaba Apache2 y Node.js directamente en el servidor, lo que provocaba que solo estuviera disponible una versión de Node.js y no pudiera ser usada para otras aplicaciones. Además, los puertos http y https estaban siendo ocupados por esta aplicación, generando una pérdida de recursos. El script recién mencionado se puede apreciar en la Fig. 18.

deploy.sh

```
#!/bin/sh

echo "Stopping apache..."
sudo systemctl stop apache2.service

echo "Stopping api..."
sudo systemctl stop timetracker-api.service

echo "Insert bitbucket username (you must have access to the timetracker repository!): "
read BITBUCKET_USERNAME

cd api/

echo "About to pull api..."
sudo git pull https://${BITBUCKET_USERNAME}@bitbucket.org/devlightsok/devlights.timetracker.api.git/ master

echo "About to install backend dependencies..."
sudo npm install

cd ../front

echo "About to pull front..."
sudo git pull https://${BITBUCKET_USERNAME}@bitbucket.org/devlightsok/devlights.timetracker.frontend.git/ master

echo "About to install frontend dependencies..."
sudo npm install

echo "About to build react app..."
sudo npm run build

cd ..

echo "Resarting api..."
sudo systemctl restart timetracker-api.service

echo "Resarting apache..."
sudo systemctl restart apache2.service

echo "Done!"
```

Fig. 18: Script de despliegue de TimeTracker.

En este proyecto, se utilizó una estructura de multi-repositorio con un repositorio para la aplicación cliente llamado “frontend” y otro para la aplicación servidor llamado “api”.
Para mejorar el uso de recursos del servidor y el proceso de despliegue de nuevas versiones de la aplicación, se decidió utilizar el repositorio de despliegues. Para ello, se hizo un *fork* del repositorio y se le cambió el nombre a "deployment" para seguir la nomenclatura utilizada en el proyecto. La estructura final quedó ilustrada en la Fig. 19.

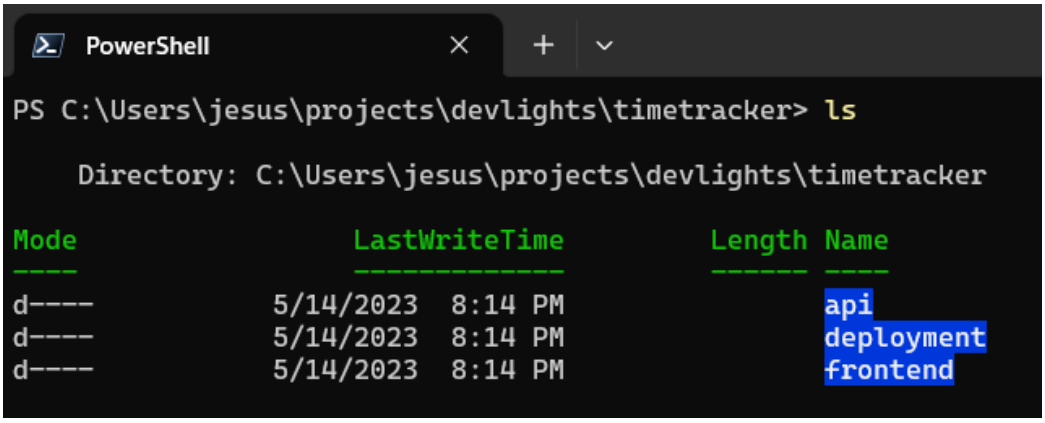


Fig. 19: Estructura del proyecto "Timetracker"

Después de editar los valores de las variables de entorno, se llevó a cabo el despliegue del repositorio en el servidor de producción de Google Cloud Platform, ejecutando los scripts correspondientes. No fue necesario redireccionar los servidores DNS, ya que el despliegue se realizó en el mismo servidor que antes, aplicando la metodología propuesta en este trabajo.

Esta aplicación se encuentra en constante desarrollo de nuevas funcionalidades, y los nuevos desarrolladores que se incorporan al equipo suelen necesitar la asistencia de los miembros más experimentados para el proceso de despliegue. Gracias a la metodología implementada, los desarrolladores que agregan nuevas características o realizan correcciones de errores pueden llevar a cabo el despliegue por sí mismos, sin necesidad de la ayuda de otros miembros del equipo. Además, el tiempo de despliegue se ha reducido significativamente, de alrededor de 30 minutos a solo 10 minutos para la primera vez que un nuevo desarrollador ejecuta el despliegue y menos de 5 minutos para las siguientes. En las Fig. 20 y Fig. 25 se presentan los gráficos correspondientes.

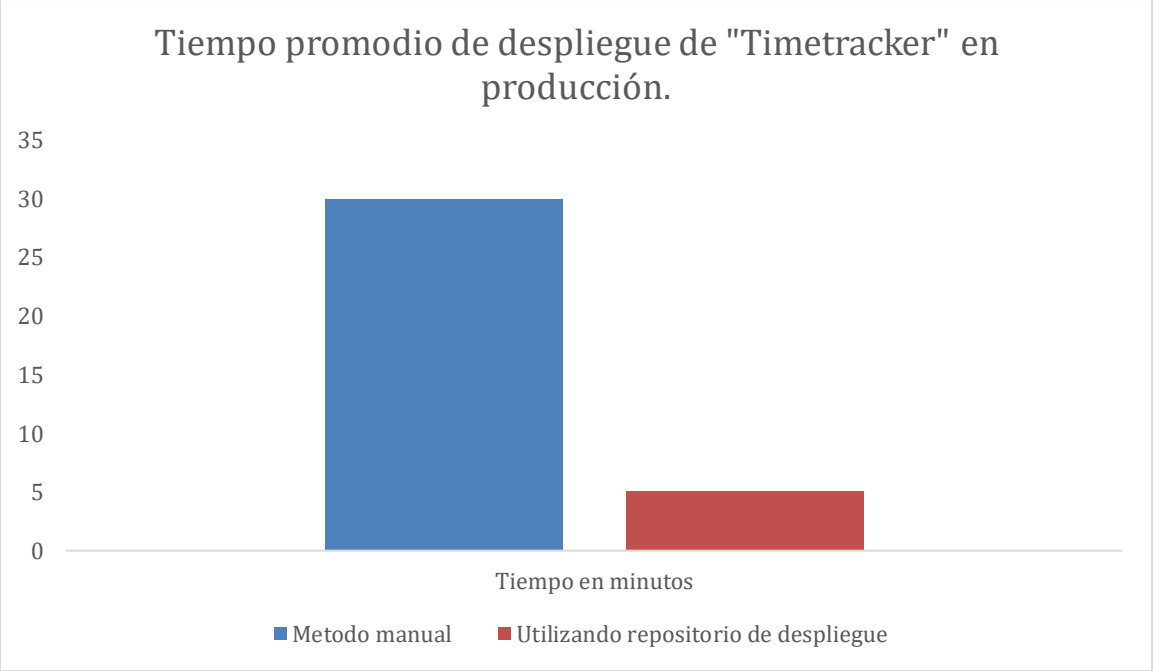


Fig. 20: Tiempo promedio de despliegue de "Timetracker" en producción.

4.3.2 Liber App (despliegue en ambiente de desarrollo y prueba)

Libera App es una aplicación web diseñada para la gestión de un estudio contable en la región. El desarrollo de esta aplicación se extendió durante un periodo de seis meses, siguiendo una estructura compuesta por una aplicación servidor (backend), una aplicación cliente (frontend) y una base de datos relacional PostgreSQL. La estructura mencionada se puede visualizar en la Fig. 21.

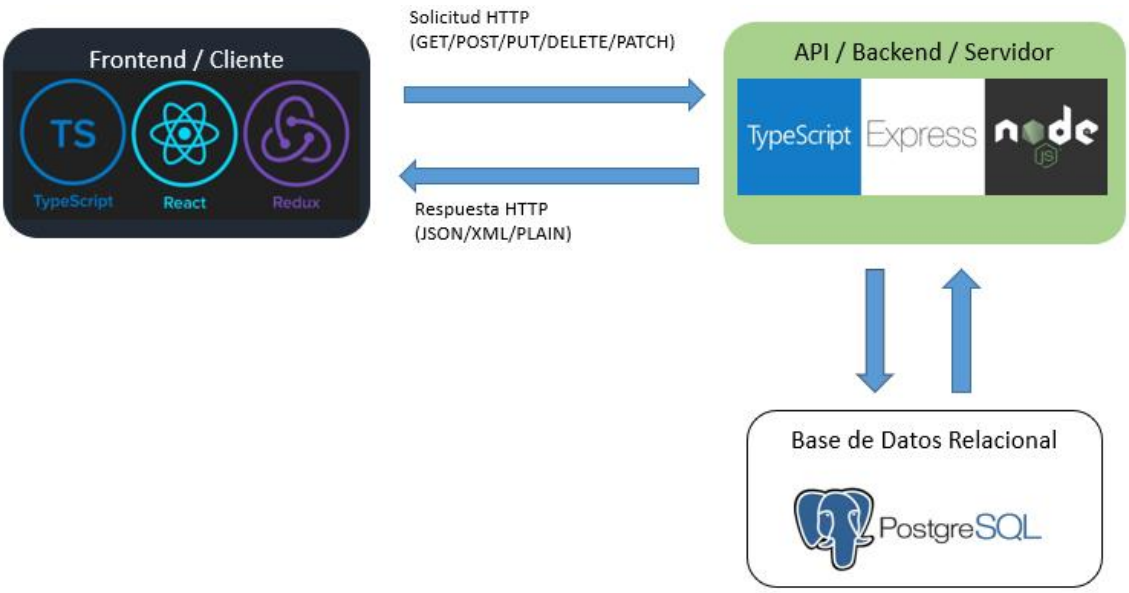


Fig. 21: Estructura de LiberApp.

Inicialmente, el proyecto Libera App estaba alojado en Bitbucket, con una aplicación cliente (liber-frontend), una aplicación servidor (liber-backend) y una base de datos relacional PostgreSQL, de acuerdo con las reglas de negocio de la empresa. Para dar actualizaciones semanales al cliente, se necesitaba realizar el despliegue manualmente cada semana, lo que resultaba tedioso y consumía mucho tiempo.
Para resolver este problema, se optó por utilizar el repositorio base de despliegue para realizar la implementación en el servidor de desarrollo y pruebas. Para ello, primero se clonó el repositorio de despliegue en el mismo nivel que los otros dos repositorios. Luego se renombró para seguir la convención que ya se estaba utilizando. Todo este proceso se puede ver en la Fig. 22.

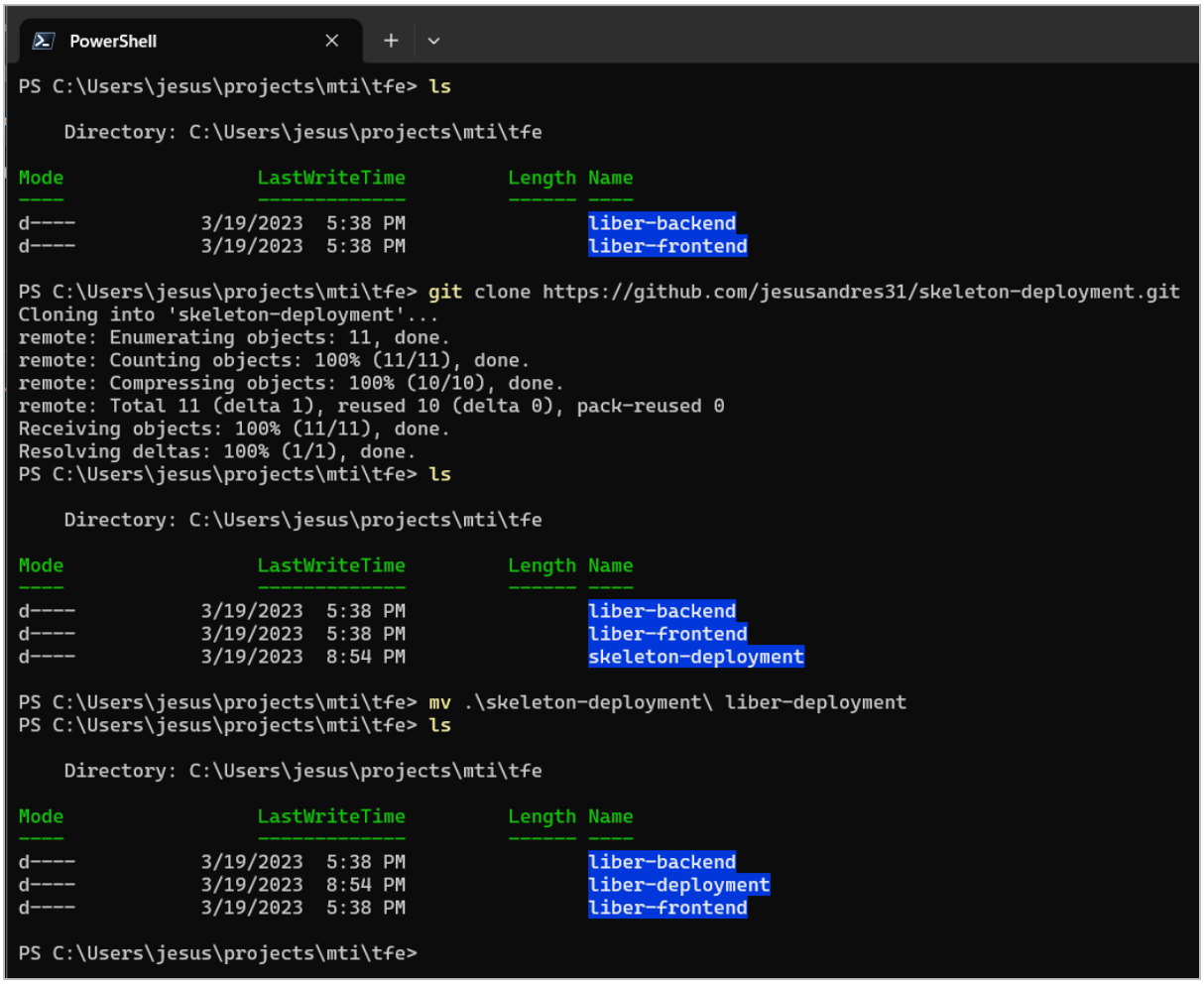


Fig. 22: Estructura del proyecto "LiberApp"

Después de configurar el repositorio base de despliegue con los valores de las variables correspondientes al proyecto, se clonó el repositorio de despliegue (liber-deployment) en el servidor de desarrollo y pruebas. La Fig. 23 muestra la estructura de contenedores en el servidor de desarrollo y pruebas.

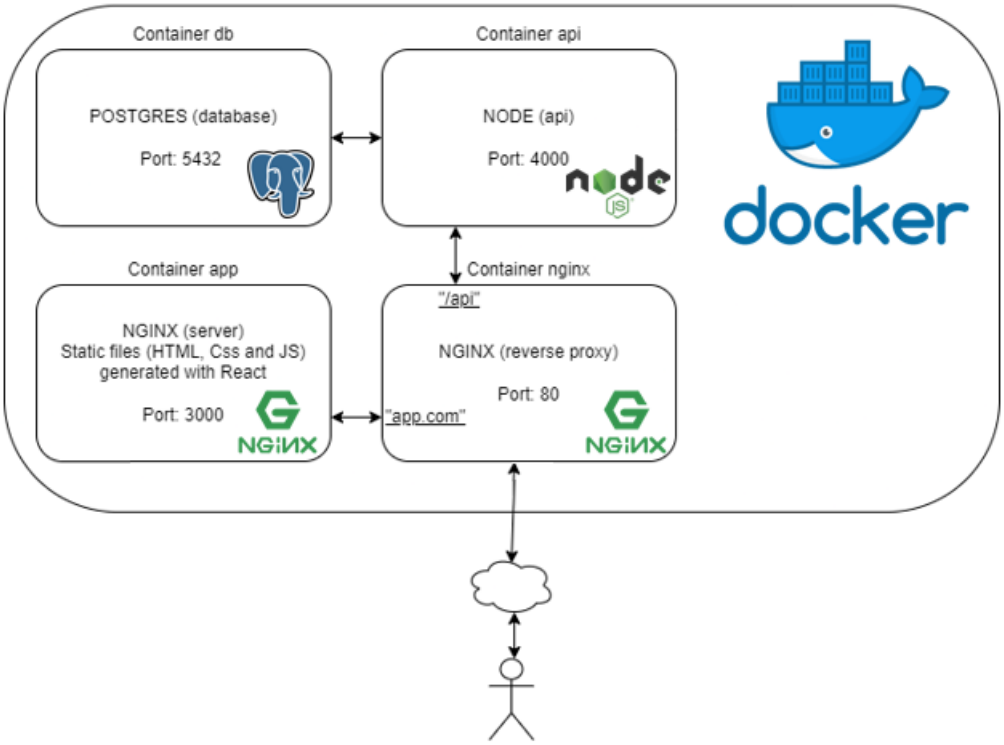


Fig. 23: Estructura de LiberApp desplegada en ambiente de prueba.

Para visualizar los contenedores en ejecución utilizamos el comando "*sudo docker ps --format "table {{.ID}}\t{{.Names}}\t{{.Ports}}" | grep liber*". La salida del comando se muestra en la Fig. 24. En detalle, se tienen los siguientes contenedores:

- **liber_app**: representa el frontend de la aplicación.
- **liber_api**: representa el backend de la aplicación.
- **liber_nginx**: es el proxy encargado de direccionar las solicitudes entrantes al frontend o al backend según corresponda.
- **Liber_db**: es la base de datos utilizada por la aplicación.

.

```
jesus_zini@dev-tech-vm:~/liber/liber-production/env.staging/deploy$ sudo docker ps --format "t
0f610935fcfc   liber_nginx              0.0.0.0:8081->80/tcp, :::8081->80/tcp
64bfecf1c9fb   liber_app                80/tcp, 3000/tcp
63fa18ff1f18   liber_api                4000/tcp
5e926ef81f5c   liber_db                 5555/tcp, 0.0.0.0:5555->5432/tcp, :::5555->5432/tcp
```

Fig. 24: Contenedores en ejecución en ambiente de desarrollo y prueba.

Gracias a esta solución, se redujo el tiempo de despliegue semanal de un promedio de 45 minutos a solo 5 a 10 minutos mediante la incorporación de automatizaciones, incluyendo la actualización automática del esquema de la base de datos en cada despliegue. En la Fig. 25 se presentan los gráficos correspondientes.

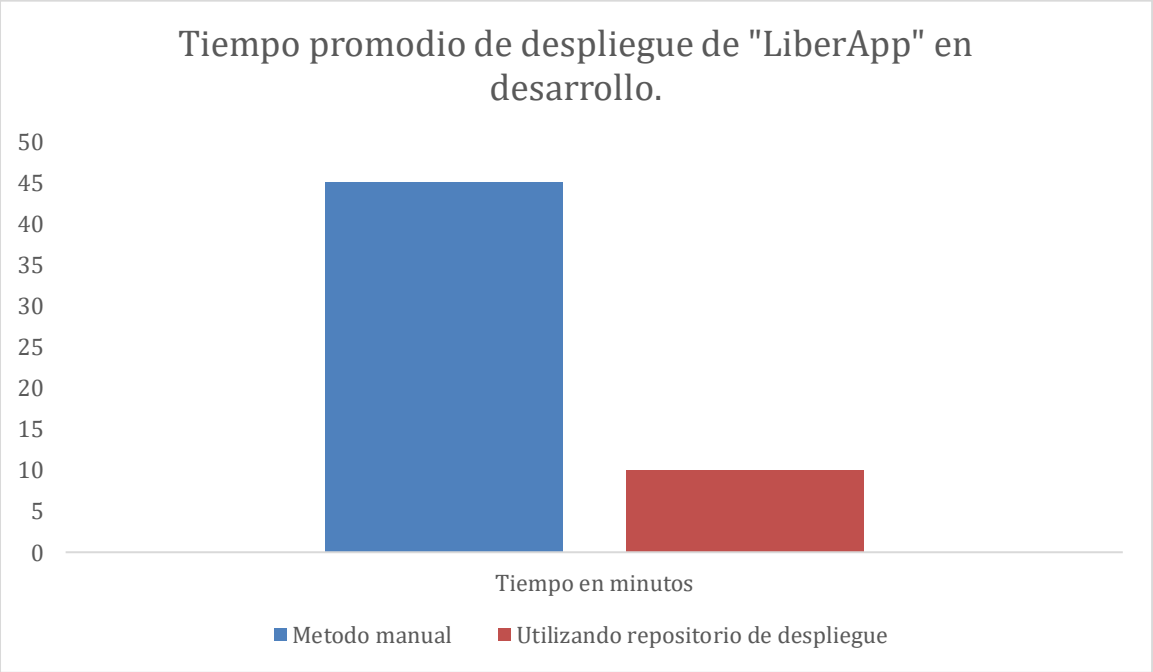


Fig. 25: Tiempo promedio de despliegue de "LiberApp" en desarrollo.

4.3.3 Liber App (despliegue en ambiente de producción)

Para el despliegue de la aplicación, se tomó la decisión de alojar todas sus partes en un solo servidor, lo que resultaba más económico y práctico. Se eligió el proveedor de servidores argentino "Donweb" para el despliegue. Se adquirió un servidor Ubuntu y se instaló Docker en él, ya que se utilizó esta herramienta para levantar los contenedores de la aplicación.

Además de los contenedores ya mencionados anteriormente, se agregaron dos más durante esta etapa de despliegue a producción. Estos contenedores se utilizan para proporcionar un certificado SSL a la aplicación y para su renovación automática. Los contenedores en cuestión son el proxy principal Nginx y el contenedor de Certbot (un servicio utilizado para generar y validar certificados SSL con la autoridad de certificación Let's Encrypt). Es importante destacar que estos dos contenedores se encuentran configurados por defecto en el repositorio de despliegue para el ambiente de producción.

Una vez que se cargó todo en el servidor, se ejecutó el script de despliegue de producción y la estructura de la aplicación desplegada se muestra en la Fig. 26.

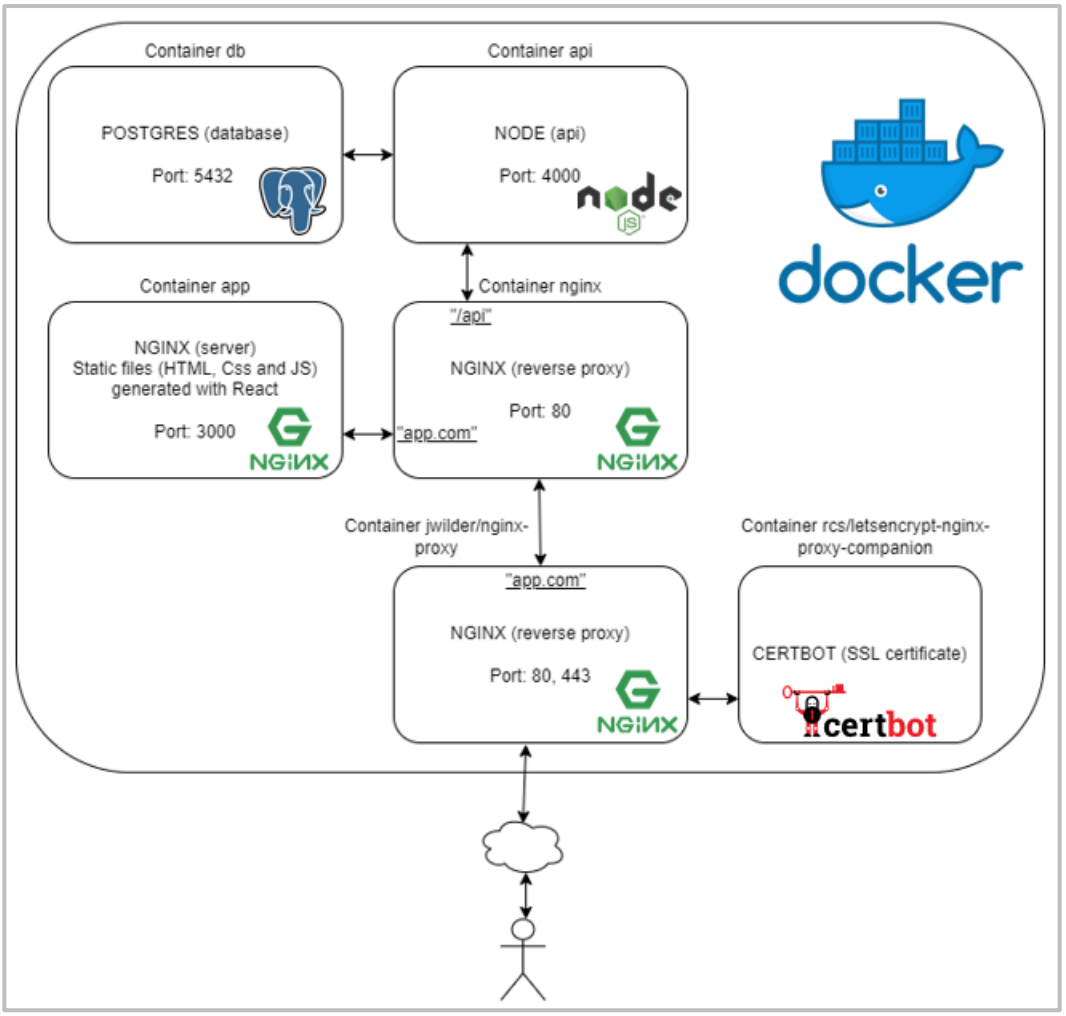


Fig. 26: Estructura de LiberApp desplegada en ambiente de producción.

Los contenedores en ejecución, que se pueden visualizar con el comando "docker ps", se encuentran representados en la Fig. 27. En detalle, se tienen los siguientes contenedores:

- **envproduction_app**: representa el frontend de la aplicación.
- **envproduction_api**: representa el backend de la aplicación.
- **envproduction_nginx**: es el proxy encargado de direccionar las solicitudes al frontend o al backend según corresponda.
- **postgres:12-alpine**: es la base de datos utilizada por la aplicación.
- **jwilder/nginx-proxy**: es el proxy principal del servidor que recibe todas las solicitudes entrantes.
- **jrcs/letsencrypt-nginx-proxy-companion**: este contenedor contiene Certbot, que se encarga de la renovación automática del certificado SSL de la aplicación.

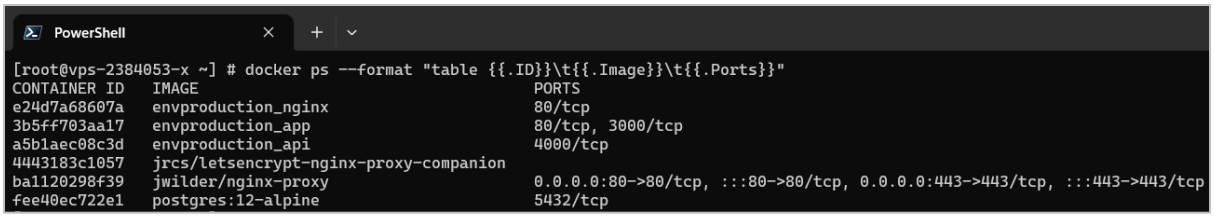


Fig. 27: Contenedores en ejecución en ambiente de producción.

Luego, teniendo en cuenta también la restauración de la base de datos de producción, el proceso de despliegue de la aplicación en el servidor de producción tardó aproximadamente una hora. Normalmente, este proceso manual lleva un día entero de trabajo (8 horas). En la Fig. 28 se presentan los gráficos correspondientes.

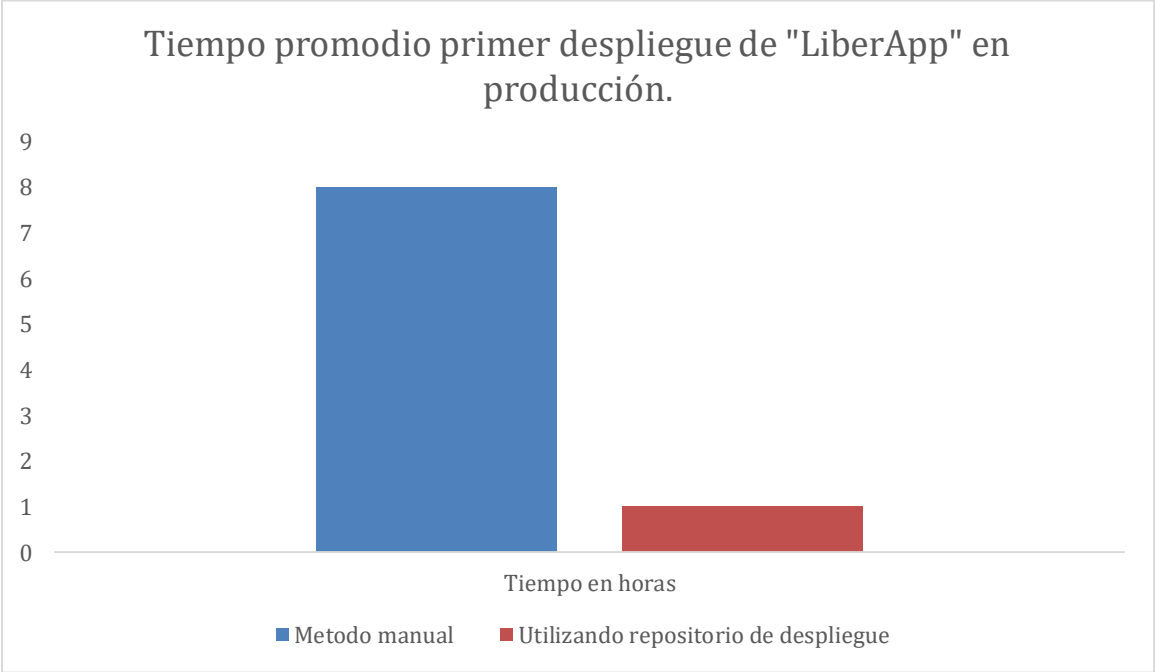


Fig. 28: Tiempo promedio de despliegue de "LiberApp" en producción.

4.3.4 Resultados finales

Como resultado de la implementación del repositorio, se obtuvo una notable reducción en el tiempo necesario para desplegar aplicaciones web locales a un servidor remoto. Esta mejora en la eficiencia del proceso de despliegue se logró gracias a la automatización de varias tareas manuales y la utilización de herramientas y scripts optimizados incluidos en el repositorio.

Para comparar los métodos de despliegue, se realizaron pruebas utilizando tanto el enfoque tradicional como el enfoque propuesto en este documento. El enfoque tradicional implica copiar el código fuente en su totalidad (tanto de la aplicación cliente como de la aplicación

servidor) directamente en el VPS, mientras que el enfoque propuesto consiste en subir únicamente el repositorio de despliegue y ejecutar los contenedores. Para evaluar la eficacia de cada método, se midieron los tiempos promedio necesarios para realizar el despliegue en ambos casos. En la Fig. 29 se presentan los gráficos que muestran los resultados promedios obtenidos en las pruebas realizadas de distintas tareas con ambos métodos.

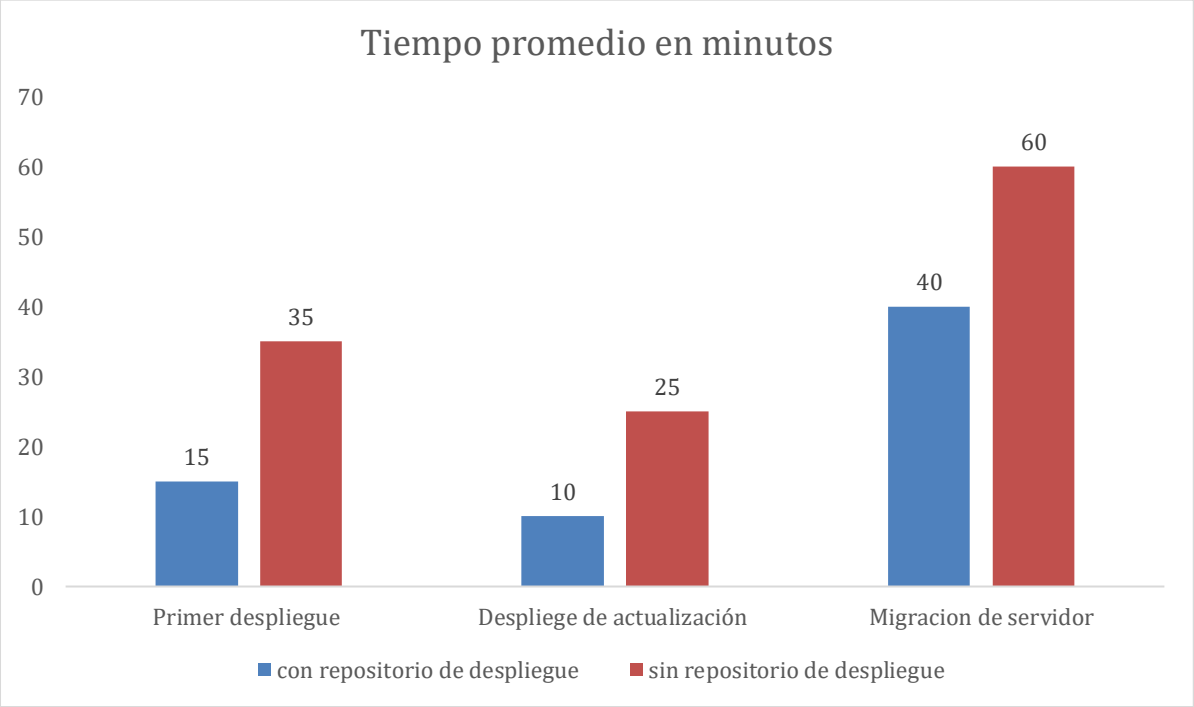


Fig. 29: Tiempo promedio de despliegue de aplicaciones web dependiendo del enfoque utilizado.

4.4 Guía para el uso del repositorio

A continuación, se presenta una guía detallada que describe el procedimiento para utilizar el repositorio de despliegue en un proyecto de aplicación web. Esta guía incluye información sobre los objetivos, alcance y los diferentes actores involucrados en el proceso.

4.4.1 Objetivos

El objetivo principal del repositorio de despliegue propuesto en este trabajo es facilitar el proceso de despliegue de aplicaciones web monolíticas en servidores VPS, reduciendo los costos y tiempos de despliegue, mejorando la eficiencia y la escalabilidad de los proyectos.

4.4.2 Alcance

El alcance del repositorio se extiende a los siguientes puntos:

- Aplicaciones web monolíticas, ya sea en un solo gran monolito que contenga toda la aplicación, o una aplicación dividida en varios servicios principales, siendo los más comunes el *backend*, *frontend* y base de datos.
- Servidores Linux, como Ubuntu, Debian, CentOS, y otros sistemas operativos similares.
- Despliegue en entornos de desarrollo o prueba.
- Despliegue en entornos de producción donde la aplicación debe coexistir con otras aplicaciones desplegadas en el mismo servidor.

4.4.3 Roles intervinientes

Los roles que intervienen en el proceso pueden variar desde equipos de 1 a N participantes, siendo los más destacados son los siguientes:

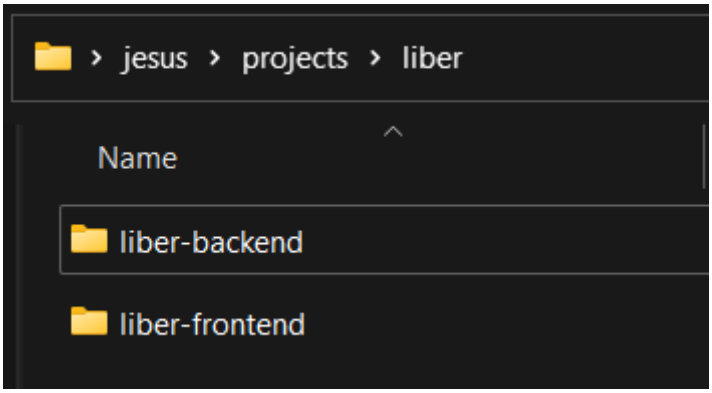
- Desarrollador: Puede tener roles tanto en el *backend* como en el *frontend*. Es responsable de la implementación y gestión de la lógica del servidor, la interacción con la base de datos y otros servicios, así como de la creación de la interfaz de usuario y la implementación de la lógica del cliente. Además, también puede desempeñar la función de desplegar la aplicación en los entornos de prueba, test y producción.
- Líder Técnico: Encargado de administrar las plataformas a utilizarse para desplegar la aplicación. Responsable de supervisar el correcto funcionamiento de los componentes desplegados en los entornos de prueba, test y producción, y asegurarse de que todos estén configurados adecuadamente.

Es importante destacar que un desarrollador del equipo puede asumir uno, varios o todos los roles pertinentes a un equipo de desarrollo de software. Además, el repositorio está diseñado para ser utilizado y administrado por todos los roles del equipo, proporcionando una solución integral y colaborativa para el despliegue de la aplicación.

4.4.4 Pasos a seguir

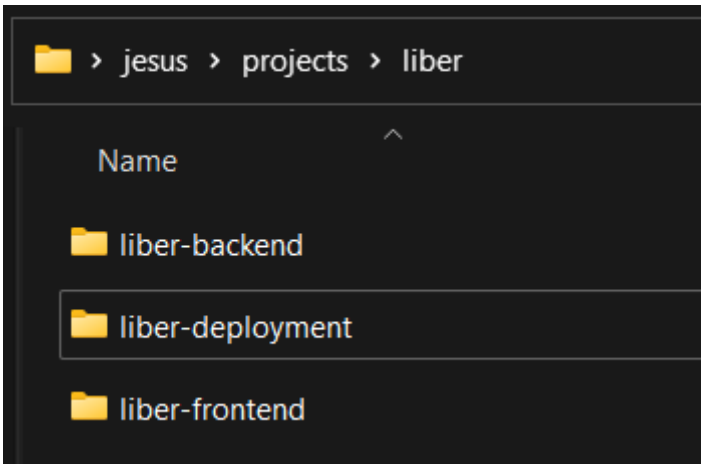
A continuación, se presenta una guía de pasos a seguir para utilizar el repositorio de despliegue de manera efectiva y eficiente.

1) El equipo dispone de un proyecto web monolítico dividido en aplicación cliente (*frontend*) y aplicación servidor (*backend*).

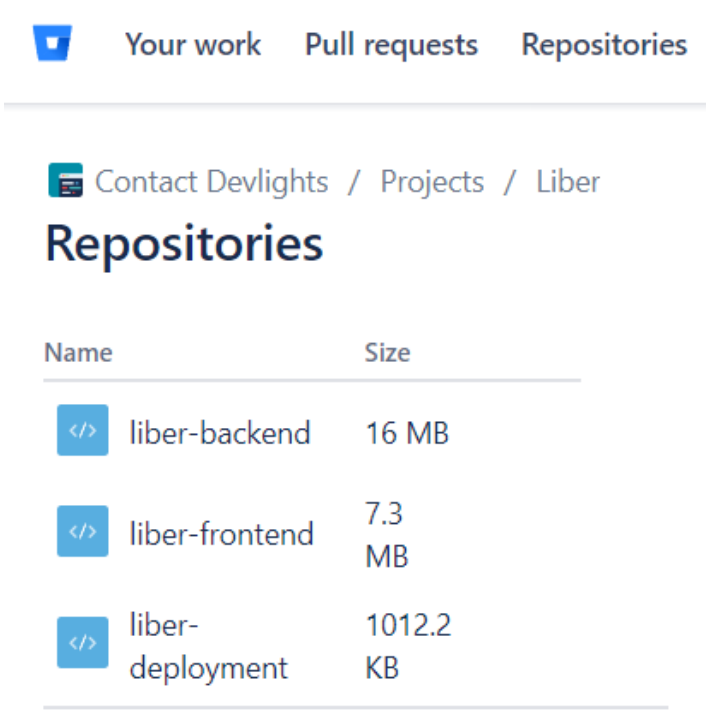


2) El desarrollador se dirige a la página del repositorio de despliegue creado en este trabajo final de especialización (https://github.com/jesusandres31/skeleton-deployment).

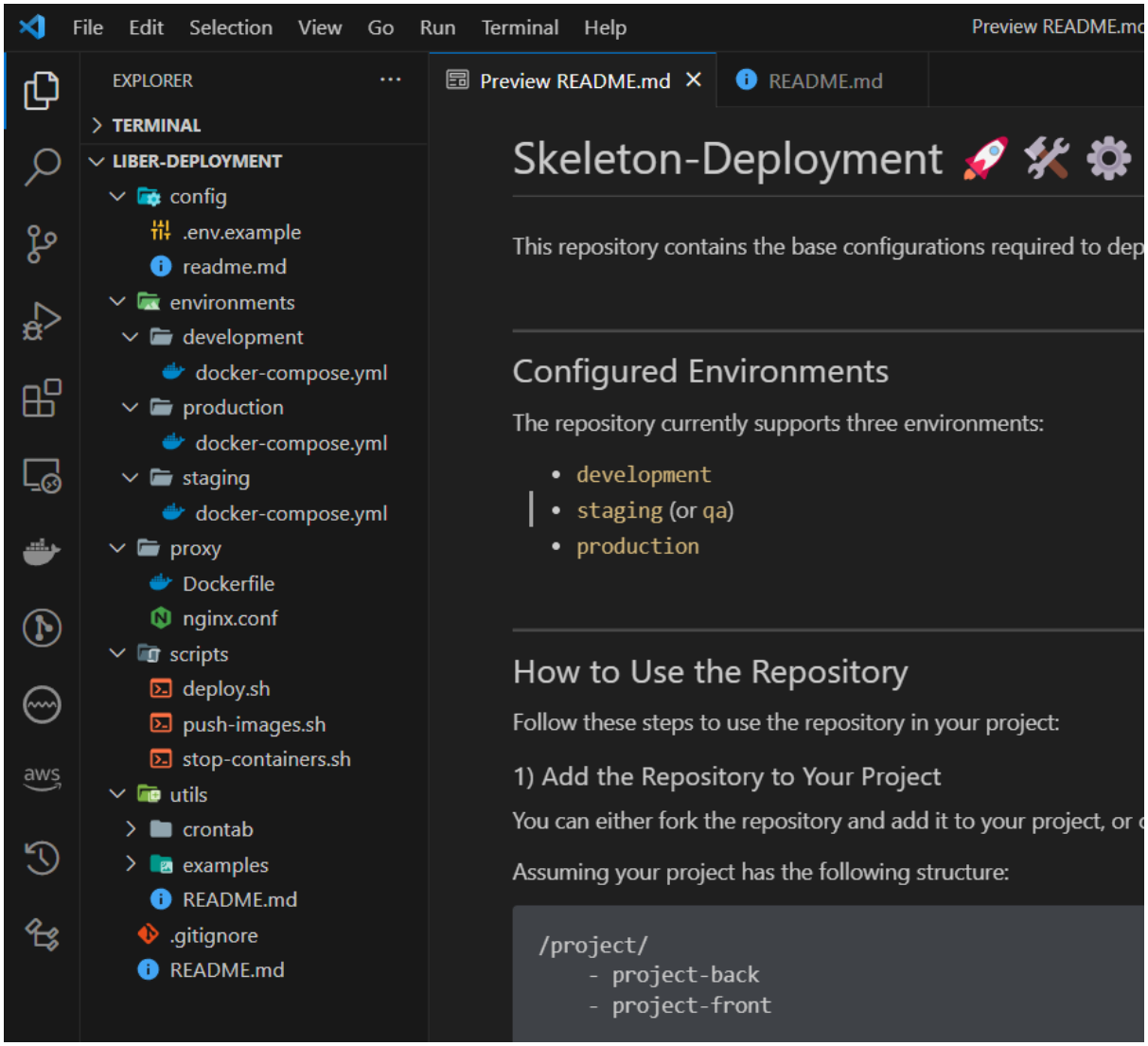
3) El desarrollador clona el repositorio en el directorio raíz del proyecto, y le cambia el nombre por uno acorde al nombre del proyecto. Luego elimina la carpeta ".git" dentro del repositorio de despliegue.



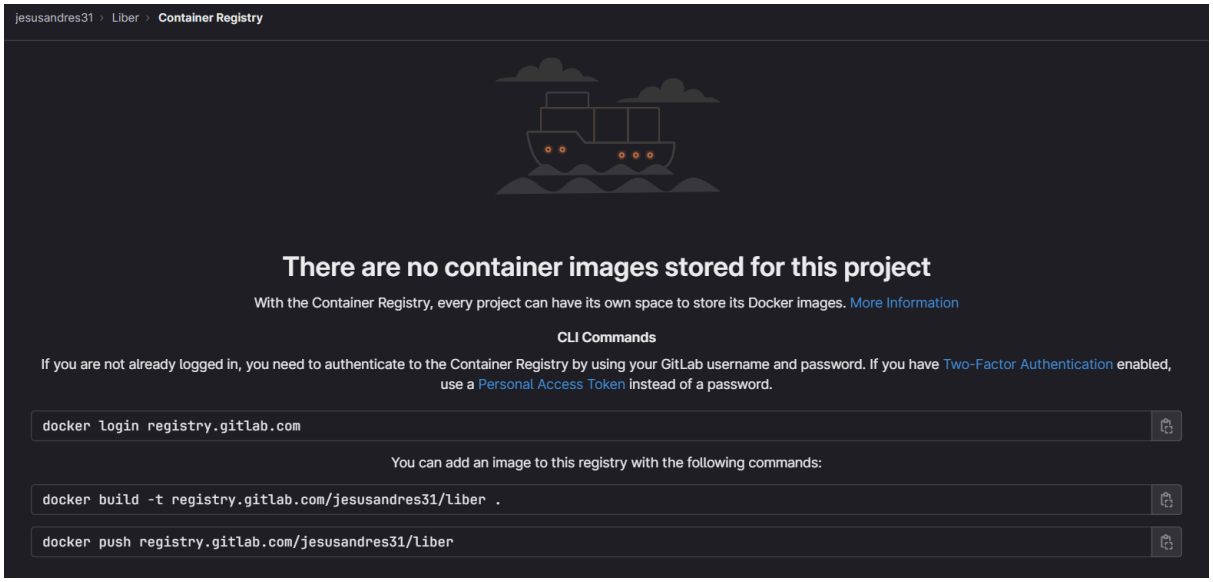
4) El líder técnico crea un repositorio con el nombre del repositorio de despliegue del proyecto en la plataforma de alojamiento de código fuente que se utiliza (Github, Gitlab, Bitbucket, etc),



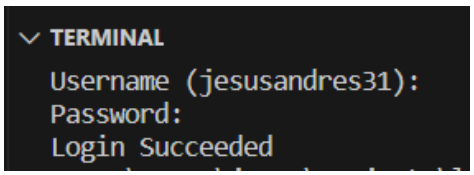
5) El desarrollador abre el repositorio de despliegue en su computador con un editor de código y sigue las instrucciones detalladas en el archivo README.md para configurar el mismo según las necesidades específicas del proyecto.



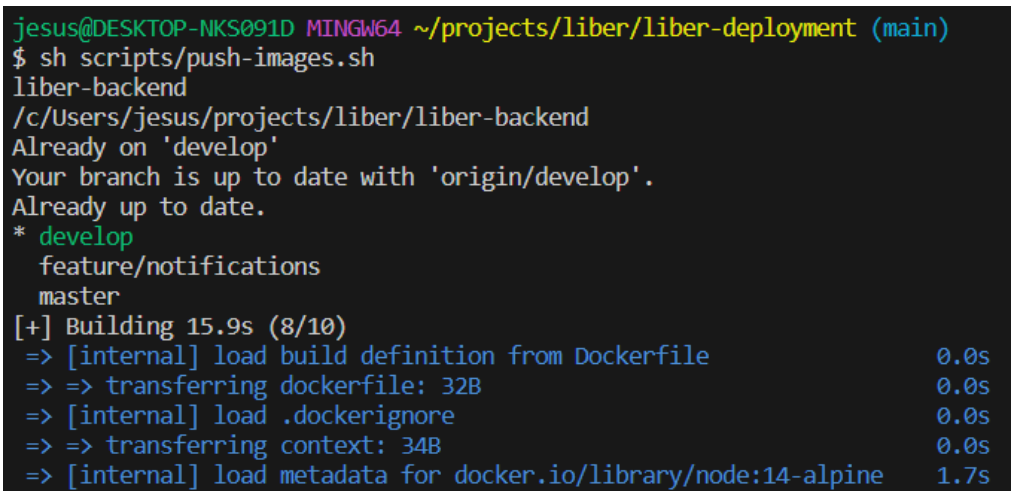
6) Entre las instrucciones del proyecto se encuentran los pasos para crear y registrar las imágenes del proyecto en un Container Registry. En este ejemplo, el equipo utiliza “Gitlab Container Registry”.



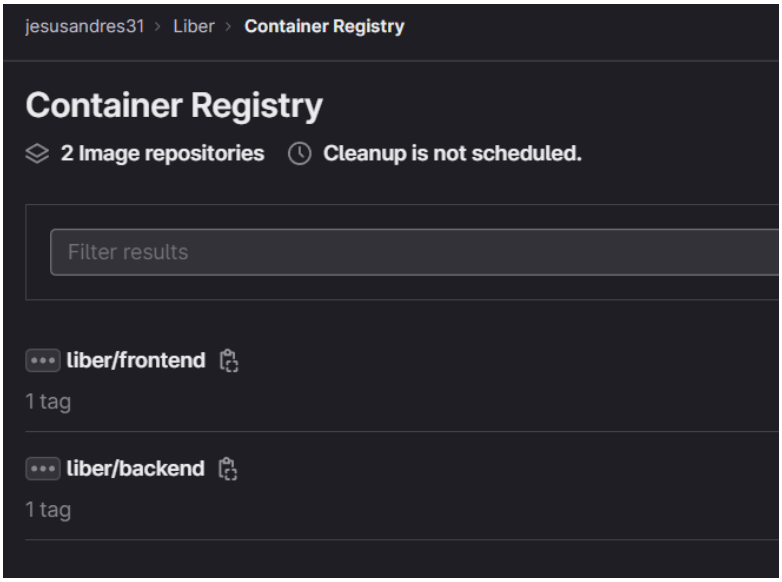
7) El desarrollador inicia sesión de la cuenta del Container Registry en la consola de su computador.



8) El desarrollador ejecuta el script que sube las imágenes del proyecto al Container Registry (*push-images.sh*). En este caso se sube el *backend* y el *frontend*.



9) El líder técnico verifica que las imágenes se encuentren ahora en la web.



10) Luego de configurar el proyecto, el desarrollador sube los cambios al repositorio de despliegue.

11) El desarrollador clona el repositorio de despliegue e inicia sesión de la cuenta del Container Registry donde están las imágenes del proyecto en el servidor VPS donde se desplegará el proyecto.

```
poli@poliserv:~/projects$ git clone https://github.com/jesusandres31/liber-deployment.git
Cloning into 'liber-deployment'...
remote: Enumerating objects: 44, done.
remote: Counting objects: 100% (44/44), done.
remote: Compressing objects: 100% (27/27), done.
remote: Total 44 (delta 10), reused 42 (delta 8), pack-reused 0
Receiving objects: 100% (44/44), 7.64 KiB | 1.53 MiB/s, done.
Resolving deltas: 100% (10/10), done.
```

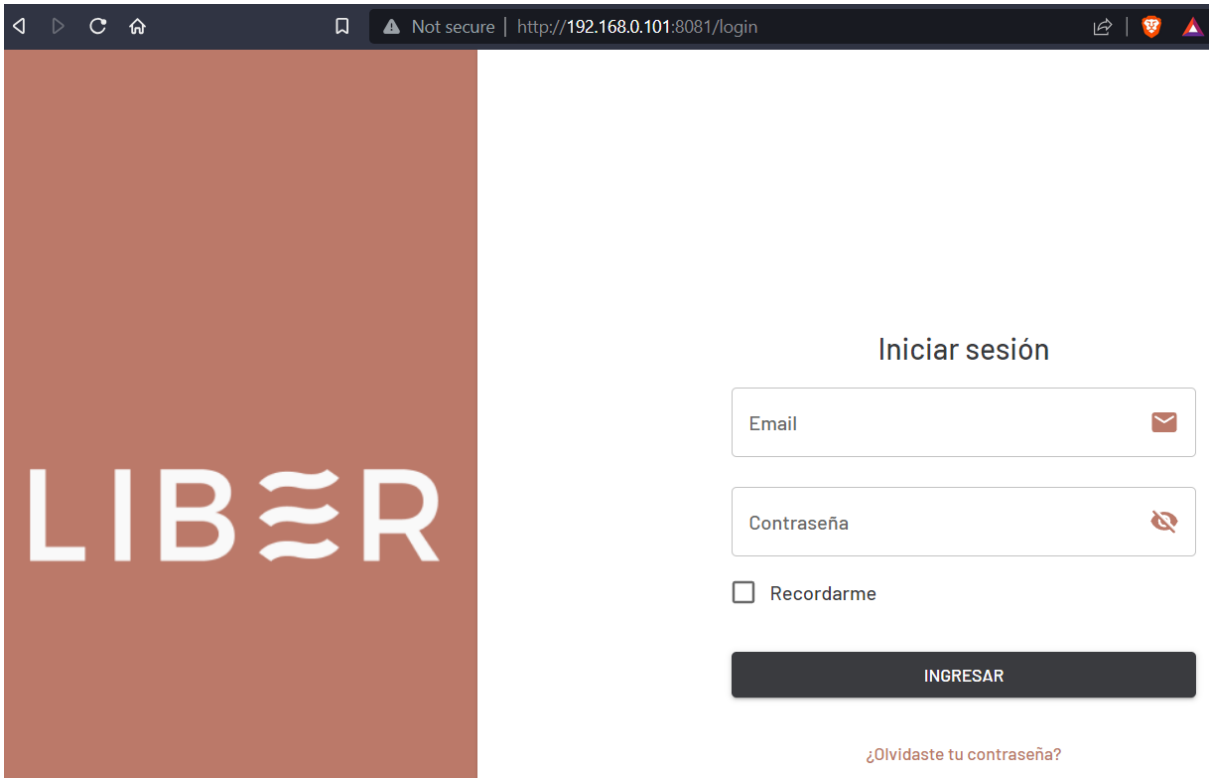
```
poli@poliserv:~/projects/liber/liber-deployment$ docker login registry.gitlab.com
Username: jesusandres31
Password:
```

12) En este punto, el desarrollador ejecuta el script de despliegue (*deploy.sh*).

```
[+] Running 5/5
 ⠿ Network staging_default  Created
 ⠿ Container liber_db       Started
 ⠿ Container liber_api      Started
 ⠿ Container liber_app      Started
 ⠿ Container liber_nginx    Started
Done!
```

13) El líder técnico verifica que los contenedores estén en ejecución y que se puede acceder remotamente a la aplicación web.

```
poli@poliserv:~/projects/liber/liber-deployment$ docker ps | grep liber
5077d85bb0ed   staging_nginx                                          "/docker-entrypoint.…"   3 minutes ago   Up 3 minutes
                                                                                                liber_nginx
6862f9c8a2e1   registry.gitlab.com/jesusandres31/liber/frontend:latest   "/docker-entrypoint.…"   3 minutes ago   Up 3 minutes
                                                                                                liber_app
e898eb7a974a   registry.gitlab.com/jesusandres31/liber/backend:latest    "docker-entrypoint.s…"   3 minutes ago   Up 47 seconds
                                                                                                liber_api
5685f86c38fc   postgres:12-alpine                                     "docker-entrypoint.s…"   3 minutes ago   Up 3 minutes
→5432/tcp                                                                                       liber_db
```



14) Además, el desarrollador puede detener la aplicación ejecutando el script correspondiente (*stop-containers.sh*).

```
poli@poliserv:~/projects/liber/liber-deployment$ sh scripts/stop-containers.sh -s
Stop STAGING env...
Stopping containers:
[+] Running 5/5
 ⠿ Container liber_nginx    Removed
 ⠿ Container liber_app      Removed
 ⠿ Container liber_api      Removed
 ⠿ Container liber_db       Removed
 ⠿ Network staging_default  Removed
Done!
```

15) Luego, para desplegar una nueva versión de la aplicación, el desarrollador sube las imágenes actualizadas al Container Registry y ejecuta nuevamente el script de despliegue en el servidor.

4.4.5 Consideraciones adicionales

Al utilizar el repositorio, es importante tener en cuenta que tiene la flexibilidad de editar y personalizar los archivos según las necesidades específicas de su proyecto. La idea principal es utilizar el repositorio como base para el proceso de despliegue, almacenando los archivos relevantes y documentando el proceso para que sirva como una guía para todos los miembros del equipo. No es necesario utilizar todos los archivos compartidos, sino adaptarlos a las necesidades particulares de su proyecto. La estructura y elementos del repositorio pueden ser modificados para que se ajusten a su aplicación.

5. Conclusiones y trabajos futuros

En este capítulo, se presenta la conclusión de este trabajo final de especialización. Se evalúa el cumplimiento de los objetivos establecidos y se resumen los logros obtenidos. Además, se mencionan posibles direcciones para futuras investigaciones.

5.1 Conclusiones

En conclusión, esta investigación ha abordado los desafíos del despliegue de aplicaciones web en equipos pequeños de desarrollo de software. Se ha analizado y seleccionado buenas prácticas de la industria del software, y se ha construido de forma iterativa e incremental, un procedimiento para desplegar aplicaciones web con arquitectura monolíticas en servidores privados virtuales, que se basó en el uso de un repositorio centralizado. Este repositorio almacena todos los archivos necesarios para el proceso de despliegue, incluyendo la documentación asociada. De esta manera, se facilita el acceso y la gestión de los recursos relacionados con el despliegue de la aplicación.

Al explorar diferentes opciones de despliegue, es importante evaluar tanto las soluciones más innovadoras, como la computación en la nube y los servicios "Serverless", como también las alternativas tradicionales basadas en servidores tradicionales. Estas últimas siguen siendo viables en muchos casos, especialmente durante las etapas de desarrollo y en situaciones con recursos limitados.

La implementación de soluciones como la propuesta en este trabajo permite reducir costos y tiempos de despliegue, mejorar la eficiencia y escalabilidad de los proyectos. Además, adoptar un enfoque basado en el uso de un repositorio centralizado para almacenar archivos y documentación relacionados con el proceso de despliegue, facilita las tareas para todos los miembros del equipo.

Si bien la propuesta concreta de este estudio puede no ser aplicable en todos los casos en específico, se recomienda al menos considerar la adopción de un estándar o marco de trabajo para el despliegue de las aplicaciones en desarrollo que involucre la documentación y el almacenamiento de archivos relevantes para dicho proceso.

5.2 Trabajos Futuros

Aunque la investigación realizada en este trabajo resulta en una solución efectiva y sencilla para el despliegue de aplicaciones en servidores VPC utilizando un repositorio de despliegue con configuraciones base, existen áreas que pueden ser mejoradas y en las que se puede trabajar en el futuro.

Entre las posibles tareas más rápidas y sencillas de mejorar se encuentran aquellas relacionadas con la optimización de procesos y la eliminación de obstáculos que puedan retrasar o dificultar el trabajo:

1) Ampliar el alcance de la solución para soportar una mayor variedad de lenguajes y tecnologías, para hacerla más versátil y adaptable a diferentes tipos de aplicaciones. Esto se puede lograr creando más archivos Dockerfile de ejemplo en la carpeta de ejemplos del repositorio.
2) Agregar un contenedor más a la estructura que sirva como monitor y que envíe alertas si algún contenedor se cae. Esto automatizaría el proceso de monitoreo post despliegue de una aplicación.
3) Agregar un script o tarea programada (*scheduler*) para el *backup* automático de la base de datos.

Luego, una limitación al utilizar este repositorio se encuentra al tratar de utilizar servicios de integración y despliegue continuo, como ser Jenkins, GitHub Actions o GitLab CI/CD. Esto se debe a que en los archivos docker-compose siempre se hace "*pull*" de las imágenes de un registro de contenedores, es decir, se espera que las imágenes ya estén subidas. Esto se podría corregir mediante la modificación de los scripts de bash y archivos Dockerfile para permitir la construcción (*build*) y subida de las imágenes en el propio servidor.

Por último, basándonos en los resultados y conclusiones obtenidos en este trabajo, y llevando a cabo un análisis más detallado acerca del despliegue de aplicaciones web en general en el mundo del desarrollo de software, y teniendo en cuenta no solo a PYMEs, sino también a empresas y aplicaciones de todos los tamaños y estructuras, se presentan algunas ideas/proyectos de mayor escala que podrían ser propuestos:

1) Se propone la creación de una aplicación que se instalaría en el entorno de VPC con el objetivo de controlar y monitorear el estado de todos los contenedores en ejecución en el servidor. Esta herramienta, instalada en el propio servidor, establecería una comunicación a través de la API de Docker con la instancia de Docker en funcionamiento, transmitiendo los datos a una aplicación frontend. Desde dicha aplicación, los desarrolladores tendrían acceso a una interfaz gráfica que les permitiría visualizar tanto el estado actual como los registros de los contenedores en el entorno VPC. Este enfoque resulta útil tanto durante el proceso de despliegue, al permitir la visualización de los registros de los contenedores, como en etapas posteriores, al facilitar el monitoreo continuo del estado de los mismos.

2) Se puede establecer/proponer una metodología de despliegue y documentación que se base en la utilización de un repositorio donde se almacenen todos los archivos relacionados al proceso de despliegue de la aplicación. Esto independientemente del proyecto, la estructura del mismo, el tipo de servidor o servidores utilizados, etc.

Referencias

[1]. N. Tankovi´c et al. “Transforming Vertical web Applications Into Elastic Cloud Applications”. 2015. IEEE

[2]. “Site Reliability Engineering”. [Online]. Available: https://sre.google/sre-book/introduction/

[3]. "A Lightweight Guide to the Theory and Practice of Scrum”. Version 2.0. 2012 Pete Deemer, Gabrielle Benefield, Craig Larman, Bas Vodde

[4]. “What is Git?”. [Online]. Available: https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F

[5]. A. M. Joy. " Performance Comparison Between Linux Containers and Virtual Machines”. International Conference on Advances in Computer Engineering and Applications (ICACEA). Ghaziabad, India. 2015

[6]. “Docker: Run your app in production”. [Online]. Available: https://docs.docker.com/get-started/orchestration/

[7]. Y. Li, & Y. Xia. “Auto-scaling web applications in hybrid cloud based on Docker”. 5th International Conference on Computer Science and Network Technology. 2016

[8]. A. Dagnino. " An Evolutionary Lifecycle Model with Agile Practices for Software Development at ABB”. ABB US Corporate Research Center. Raleigh, NC, USA. 2002

[9]. “Patterns for Managing Source Code Branches”. [Online]. Available: https://martinfowler.com/articles/branching-patterns.html

[10]. “What is a Container?”. [Online]. Available: https://www.docker.com/resources/what-container

[11]. “LXC: Linux Containers”. [Online]. Available: https://linuxcontainers.org/

[12]. “Podman Container”. [Online]. Available: https://podman.io/

[13]. “Container and virtualization tools”. [Online]. Available: https://sre.google/sre-book/introduction/

[14]. “What is Docker Hub?”. [Online]. Available: https://www.docker.com/products/docker-hub/

[15]. “GitLab Container Registry”. [Online]. Available: https://docs.gitlab.com/ee/user/packages/container_registry/

[16]. J. Shah, & D. Dubaria. “Building Modern Clouds: Using Docker, Kubernetes & Google Cloud Platform”. IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC). 2019

[17]. M. Villamizar et al. “Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy web Applications in the Cloud”. 2015. IEEE

[18]. “What is Infrastructure as Code (IaC)?”. [Online]. Available: https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac

[19]. B. Piedade, J. P. Dias & F. F. Correia. "An Empirical Study on Visual Programming Docker Compose Configurations". Faculty of Engineering, University of Porto INESC TEC Porto, Portugal. 2020

[20]. M. Virmani. "Understanding DevOps & Bridging the gap from Continuous Integration to Continuous Delivery". Fifth international conference on Innovative Computing Technology (INTECH 2015). 2015

[21]. M. Shahin, M. A. Babar & L. Zhu. "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices". Faculty of Engineering, University of Porto INESC TEC Porto, Portugal. 2020

[22]. "CapRover". [Online]. Available: https://caprover.com/docs/get-started.html

[23]. J. Fischer, R. Majumdar & S. Esmaeilsabzali. "Engage: A Deployment Management System". PLDI'12. Beijing, China. 2012

[24]. Y. Luo. " An Open-Source and Portable MLOps Pipeline for Continuous Training and Continuous Deployment". Faculty of Science. University of Helsinki. Abril 2, 2023.

[25]. "Kubernetes". [Online]. Available: https://kubernetes.io/

[26]. "Helm". [Online]. Available: https://helm.sh/

[27]. "Linkerd". [Online]. Available: https://linkerd.io/

[28]. "Istio". [Online]. Available: https://istio.io/

[29]. "Cloudformation". [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html

[30]. "Terraform". [Online]. Available: https://www.terraform.io/

[31]. "Apache Kafka". [Online]. Available: https://kafka.apache.org/

[32]. "RabbitMQ". [Online]. Available: https://www.rabbitmq.com/

[33]. "CertBot". [Online]. Available: https://certbot.eff.org/

[34]. "Let's Encrypt". [Online]. Available: https://letsencrypt.org/

[35]. "Promoting Jira configuration from development to production". [Online]. Available: https://confluence.atlassian.com/adminjiraserver/promoting-jira-configuration-from-development-to-production-1167739614.html