

UNIVERSIDAD DEL VALLE DE GUATEMALA  
Facultad de Ingeniería



**Automatización de *deployment* de plataformas web que permita a las  
empresas tener lanzamientos más frecuentes y confiables**

Trabajo de graduación presentado por Diana Ximena de León Figueroa  
para optar al grado académico de Licenciada en Ingeniería en Ciencia de la  
Computación y Tecnologías de la Información

Guatemala,

2022







UNIVERSIDAD DEL VALLE DE GUATEMALA  
Facultad de Ingeniería



**Automatización de *deployment* de plataformas web que permita a las  
empresas tener lanzamientos más frecuentes y confiables**

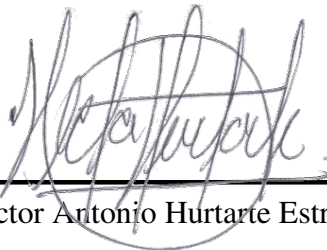
Trabajo de graduación presentado por Diana Ximena de León Figueroa  
para optar al grado académico de Licenciada en Ingeniería en Ciencia de la  
Computación y Tecnologías de la Información

Guatemala,

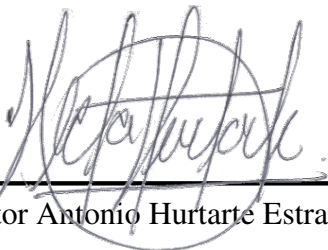
2022



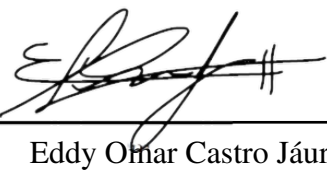
Vo.Bo.:

(f)   
Héctor Antonio Hurtarte Estrada

Tribunal Examinador:

(f)   
Héctor Antonio Hurtarte Estrada

(f)   
Andrea Adriana Grimaldi Santos

(f)   
Eddy Omar Castro Jáuregui

Fecha de aprobación: Guatemala, 7 de diciembre de 2022.





---

## Prefacio

---

Como parte de un equipo de desarrollo, he vivido de primera mano las complejidades que se presentan para la creación de instancias y actualización de plataformas web (deployment). Este proyecto busca facilitar esa fase del proceso de desarrollo de plataformas web brindando los conocimientos y herramientas necesarias para que pueda ser implementado por las empresas dedicadas al desarrollo de software en Guatemala.

Agradezco grandemente a cada persona que me ha acompañado en este camino universitario. Iniciando por mi familia que me ha apoyado desde el inicio, Don Carlos Paiz Andrade por la confianza y el apoyo brindando mediante una beca de estudio. También a mis amigos, compañeros de clase y profesores que sin duda me han ayudado a llegar hasta aquí.



<b>Prefacio</b>	V
<b>Lista de figuras</b>	IX
<b>Lista de cuadros</b>	XI
<b>Resumen</b>	XIII
<b>Abstract</b>	XV
<b>1. Introducción</b>	1
<b>2. Antecedentes</b>	3
<b>3. Justificación</b>	5
<b>4. Objetivos</b>	7
4.1. Objetivo general	7
4.2. Objetivos específicos	7
<b>5. Marco teórico</b>	9
5.1. SaaS	9
5.1.1. ¿Cómo funciona SaaS?	9
5.1.2. Ventajas	10
5.1.3. Inconvenientes	11
5.1.4. Tipos	11
5.2. DevOps	12
5.2.1. Principios de DevOps	12
5.2.2. Prácticas de DevOps	12
5.3. Control de versiones	13
5.3.1. Subversion	13
5.3.2. Git	14
5.4. Pruebas automatizadas	14
5.5. Metodologías de CI/CD	15

5.5.1. ¿Cuál es la diferencia entre CI y CD en el concepto CI/CD?	15
5.5.2. Beneficios de la implementación de CI/CD	15
5.5.3. Desafíos	16
5.5.4. Herramientas	16
5.6. ¿Qué es un contenedor de Docker?	19
5.6.1. Terminología de Docker	19
<b>6. Marco metodológico</b>	<b>21</b>
6.1. Análisis de la situación actual	21
6.2. Evaluación y realización de pruebas automatizadas	23
6.3. Deployer	24
6.3.1. Implementación	25
6.4. Gitlab CI/CD	34
6.4.1. Migrar de SVN a GitLab	35
6.4.2. Configuración del servidor	36
6.4.3. Crear una imagen de contenedor	37
6.4.4. Configuración del registro de contenedores de GitLab	38
6.4.5. Configuración de Gitlab CI/CD	39
6.5. Integración con plataformas de notificación y manejo de errores	43
6.5.1. GitLab + Sentry	43
6.5.2. GitLab + Slack	43
6.5.3. Deployer + Sentry + Slack	43
<b>7. Resultados</b>	<b>45</b>
7.1. Pipelines	45
7.1.1. Métricas de GitLab Analytics	46
7.2. Mediciones de tiempo utilizando GitLab CI/CD y Deployer	46
<b>8. Análisis de resultados</b>	<b>49</b>
<b>9. Conclusiones</b>	<b>51</b>
<b>10. Recomendaciones</b>	<b>53</b>
<b>11. Referencias de la red o de internet</b>	<b>55</b>
<b>12. Anexos</b>	<b>57</b>
12.1. Creación de una cultura de DevOps	57
12.2. Herramientas de pruebas automatizadas	58

---

## Lista de figuras

---

1. Diferencia entre las metodologías de CI/CD.	15
2. Funcionamiento de CI/CD en AWS CodePipeline.	17
3. Funcionamiento de Azure pipelines.	17
4. Funcionamiento de CI/CD con Azure pipelines.	17
5. Funcionamiento de CI/CD con Jenkins.	18
6. Funcionamiento de CI/CD con GitLab CI/CD.	18
7. Funcionamiento de CI/CD con Travis CI.	19
8. Ejecución de pruebas automatizadas.	24
9. Creación de usuario y asignación de permisos.	36
10. Generación de claves SSH.	36
11. Agregar clave privada a variables de CI/CD.	37
12. Agregar clave pública como clave de implementación.	37
13. Compilación de archivo Dockerfile.	38
14. Agregar imagen de Docker al registro.	39
15. Pipelines del proyecto	41
16. Etapas del pipeline	42
17. Etapas del pipeline	42
18. Estructura de la plataforma con Deployer	43
19. Pipelines realizadas exitosamente.	45
20. GitLab CI/CD Analytics - pipelines ejecutadas, fallidas y finalizadas correctamente.	46
21. GitLab CI/CD Analytics - gráfico de pipelines ejecutadas en la última semana.	46
22. Gráfica de área de mediciones de tiempo para la creación de nuevas instancias de la plataforma	47
23. Gráfica de caja y bigotes para mediciones de tiempo para la creación de nuevas instancias de la plataforma	47
24. Gráfica de área de mediciones de tiempo para la actualización de un conjunto de instancias de la plataforma	48
25. Gráfica de caja y bigotes para mediciones de tiempo para la actualización de un conjunto de instancias de la plataforma	48



---

## Lista de cuadros

---

1. Tiempos creación de instancias de usuario - instalación . . . . .	21
2. Tiempos creación de instancias de instalaciones . . . . .	22
3. Tiempos actualización de usuario - instalación . . . . .	22
4. Tiempos actualización de instalaciones . . . . .	23
5. Tiempos creación de instancias de instalaciones . . . . .	47
6. Tiempos actualización de instalaciones . . . . .	47
7. Tiempos promedios . . . . .	48





Las empresas enfrentan una nueva dinámica, de un mundo conectado, tecnologías accesibles a la población y la creación de experiencias digitales que establecen un nuevos estándares para la interacción con los clientes. En respuesta a esto las empresas, sin excepción, deben iniciar un proceso de transformación digital, donde el software es uno de los elementos claves.

Así pues, es importante para las empresas contar con las capacidades de crear software con criterios de velocidad, calidad y eficiencia para preservar su existencia. La metodología de DevOps particularmente tácticas como CI y CD proveen una solución al proceso de ingeniería de software de una empresa para lograr producir y mantener software bajo las exigencias antes mencionadas.

Se implementó una pipeline de integración continua y entrega continua utilizando las herramientas de GitLab CI/CD y Deployer para una plataforma web publicada en un ambiente Linux, es recomendable corroborar su funcionalidad en otros ambientes.



Companies face a new dynamic, a connected world, technologies accessible to the population and the creation of digital experiences that establish new standards for interaction with customers. In response to this, companies without exception, must start a process of digital transformation, where software is one of the key elements.

Therefore, it is important for companies to have the capabilities to evolve software with criteria of speed, quality and efficiency to preserve their existence. The DevOps methodology, particularly tactics such as CI and CD, provide a solution to the software engineering process of a company in order to produce and maintain software under the aforementioned requirements.

A continuous integration and continuous delivery pipeline was implemented using the GitLab CI/CD and Deployer tools for a web platform published in a Linux environment, it is recommended to verify its functionality in other environments.



# CAPÍTULO 1

---

## Introducción

---

Un componente clave de las metodologías ágiles son los métodos de integración continua, sus principales objetivos son encontrar y solucionar problemas potenciales más rápido y disminuir el tiempo en la creación de nuevas instancias. Un 48 % de los desarrolladores de pequeñas empresas indicó utilizar herramientas de CI/CD en su flujo de trabajo para el informe State of Continuous Delivery del 2021 (Dee, [2021](#)).

Este proyecto busca demostrar cómo la cultura de metodologías ágiles puede disminuir el tiempo que los desarrolladores utilizan en la creación de instancias y actualización de plataformas web (deployment). Se selecciona una plataforma web para utilizar como base, sin embargo, toda la configuración del pipeline puede ser modificada para adaptarse a cualquier otro caso de negocio.

Se utilizan las herramientas de GitLab CI/CD y Deployer para la implementación del pipeline, con lo cual se logra reducir en hasta 15 veces el tiempo para crear una nueva instancia de la plataforma web y en casi 3 veces el tiempo de actualización de un conjunto de instancias de la plataforma.



---

### Antecedentes

---

La plataforma web SaaS utilizada en el proyecto, tiene la característica que funciona mediante una branch versionada por medio de Subversion por cada instalación, lo que quiere decir que se actualiza y da mantenimiento a múltiples instalaciones (alrededor de 60 actualmente) y por cada nuevo cliente se debe crear un nuevo branch y crear una nueva instancia de la aplicación. El equipo de desarrollo de la plataforma utiliza Zoho Projects para el manejo y control de tiempo invertido.

La plataforma web está desarrollada en PHP, con el framework de Symfony, utilizando Propel como ORM y bases de datos con instancias en MySQL y MariaDB. Asimismo los dominios se manejan desde la registradora de dominios GoDaddy.

La creación de una nueva instancia de la plataforma web, implica tener conexión al servidor en el cual se aloja el versionamiento del código fuente, de ser una plataforma que involucre base de datos, es posible que se separe en dos servidores uno de aplicaciones y uno de bases de datos, lo que implica tener acceso a ambos servidores.

Crear la branch del repositorio de versiones y asignar los permisos necesarios según corresponda. Conjunto crear el directorio y alojar el proyecto en él, instalar librerías y dependencias necesarias para el funcionamiento de la plataforma.

En caso de que la plataforma utilice base de datos, acceder a crear la base de datos con su estructura mínima necesaria. Luego regresar al proyecto para configurar la información de acceso a la base de datos (nombre, usuario, credenciales, etc.). Verificar si la base de datos y el proyecto se encuentran al día, es decir, si no existen migraciones pendientes.

Como siguiente paso, realizar la configuración en el servidor, crear el subdominio para la plataforma web, asimismo, generar el certificado para la utilización de HTTPS, configurar el certificado en el servidor y por último reiniciar el servidor y verificar que la instalación se haya levantado correctamente.





Las empresas de desarrollo de software en Guatemala para competir a nivel internacional necesitan crear y ajustar sus tecnologías, incluyendo sus servicios y aplicaciones de software con mayor agilidad sin elevar los costos que se tienen designados para ello. De acuerdo con el informe “2021 State of Continuous Delivery” reportó que el 44 % de los desarrolladores utiliza CI o CD, asimismo alrededor del 18 % utiliza ambas prácticas para automatizar por completo sus procesos de entrega. El informe también refleja que casi el 60 % de desarrolladores empresariales usan CI/CD en su flujo de trabajo, de esos un 48 % indicó trabajar para pequeñas compañías y un 42 % afirmó que en sus proyectos freelancers CI/CD juega un papel importante (Dee, [2021](#)).

Las organizaciones que utilizan de forma efectiva DevOps tienen deployments 46 veces más frecuentes que sus competidores, recuperación de fallas 96 veces más rápida, tiempo de entrega 440 veces más rápido para los cambios (CDNNetworks, [2022](#)). Las empresas que carezcan de la capacidad de implementar o mejorar las metodologías actuales serán sobrepasadas por aquellas que hayan implementado prácticas para lograr este cometido, por tanto, cobra importancia sintetizar el conocimiento que pueda ser tomado como referencia para iniciar la transformación de los procesos actuales de entrega de software.



### 4.1. Objetivo general

Automatizar el proceso de deployment y traslado de software entre un entorno de pruebas y producción haciendo uso de un deployment pipeline que permita tener lanzamientos más frecuentes y confiables que aumenten la satisfacción de los usuarios finales sin utilizar procesos manuales.

### 4.2. Objetivos específicos

- Identificar el tiempo promedio utilizado en levantar la plataforma en producción y en realizar actualizaciones a la plataforma.
- Implementar un proceso de creación de infraestructura por medio de CI/CD para la plataforma web.
- Definir la herramienta CI/CD apropiada para la plataforma web seleccionada.
- Implementar las pruebas automatizadas necesarias para garantizar la continuidad del servicio.
- Crear pipeline y procedimientos automatizados que permitan mejorar la eficiencia para gestionar su infraestructura de TI.
- Disminuir los procesos manuales que llevan a cabo los equipos de desarrollo en el proceso de creación de instancias y actualizaciones de la plataforma web.
- Medir el nuevo tiempo de lanzamiento y actualización de la plataforma.



### 5.1. SaaS

Software como un Servicio (SaaS), es un modelo de distribución de software donde el soporte lógico y los datos que maneja se alojan en servidores de una compañía de tecnologías de información y comunicación (TIC), a los que se accede vía Internet desde un cliente. La empresa proveedora TIC se ocupa del servicio de mantenimiento, de la operación diaria y del soporte del software usado por el cliente. Regularmente el software puede ser consultado en cualquier computador, se encuentre presente en la empresa o no. Se deduce que la información, el procesamiento, los insumos y los resultados de la lógica de negocio del software están hospedados en la compañía de TIC (Wesley Chai, [2021](#)).

#### 5.1.1. ¿Cómo funciona SaaS?

En este modelo, un proveedor de software independiente puede contratar a un proveedor de nube externo para alojar la aplicación, o el proveedor de la nube también podría ser el proveedor de software. SaaS es una de las tres categorías principales de computación en la nube, junto con la infraestructura como servicio (IaaS) y la plataforma como servicio (PaaS). Una variedad de profesionales de TI, usuarios comerciales y usuarios personales utilizan aplicaciones SaaS. Los productos van desde entretenimiento personal, hasta herramientas de TI avanzadas. A diferencia de IaaS y PaaS, los productos SaaS se comercializan con frecuencias para usuarios Business to Business (B2B), Business to Consumer (B2C) (Wesley Chai, [2021](#)).

Un proveedor de software alojará la aplicación y los datos relacionados utilizando sus propios servidores, bases de datos, redes y recursos informáticos, o puede ser un Independent Software Vendor (ISV) que contrate a un proveedor de nube para alojar la aplicación en el centro de datos del proveedor. La aplicación será accesible para cualquier dispositivo con conexión a la red, por lo general, se accede a través de navegadores web. Según el informe de McKinsey & Company, los analistas de la industria de la tecnología predicen un mayor crecimiento en el mercado de software

como servicio y esperan ver el mercado de productos de SaaS cerca de \$ 200 mil millones para 2024. Las empresas que utilizan aplicaciones SaaS no tienen la tarea de configurar y mantener el software. Los usuarios simplemente pagan una tarifa de suscripción para obtener acceso al software, que es una solución lista para usar (Wesley Chai, 2021).

En el modelo SaaS de software a pedido, el proveedor brinda a los clientes acceso basado en la red a una sola copia de una aplicación que el proveedor creó específicamente para la distribución de SaaS. El código fuente de la aplicación es el mismo para todos los clientes, y cuando se lanzan nuevas características o funcionalidades, se implementan para todos los clientes. Dependiendo del Service Level Agreement (SLA), los datos del cliente para cada modelo pueden almacenarse localmente, en la nube o tanto local como en la nube. Las organizaciones pueden integrar aplicaciones SaaS con otro software utilizando Application Program Interface (API). Por ejemplo, una empresa puede escribir sus propias herramientas de software y usar las API del proveedor de SaaS para integrar esas herramientas (Wesley Chai, 2021).

### 5.1.2. Ventajas

SaaS elimina la necesidad que las organizaciones instalen y ejecuten aplicaciones en sus propias computadoras o en sus propios centros de datos. Esto elimina gastos de adquisición y mantenimiento de hardware, así como las licencias e instalación de software. No es necesario contar con un área especializada de soporte para el sistema, por lo que se reducen los costos y riesgo de inversión. Otros beneficios del modelo SaaS incluyen:

- No es necesaria la compra de una licencia para utilizar el software, sino el pago de un alquiler o renta por el uso del software. La transición de los costos a un gasto operativo recurrente permite que muchas empresas ejerzan una presupuestación mejor y más predecible. Aunque también se dan casos particulares donde el servicio es totalmente gratuito, es decir, se cuenta con el servicio, se puede acceder libremente, se garantiza usabilidad y actualidad, pero no se paga por el servicio.
- La responsabilidad de la operación recae en la empresa IT. Esto significa que la garantía de disponibilidad de la aplicación y su correcta funcionalidad es parte del servicio que da la compañía proveedora del software.
- En lugar de comprar software nuevo, los clientes pueden confiar en un proveedor de SaaS para realizar actualizaciones y administrar parches automáticamente. La empresa IT no desatiende al cliente. El servicio y atención continua del proveedor al cliente es necesaria para que este último siga pagando el servicio.
- La empresa IT provee los medios seguros de acceso en los entornos de la aplicación. Si una empresa IT quiere dar SaaS en su cartera de productos, debe ofrecer accesos seguros para que no se infiltren datos privados en la red pública.
- Los servicios SaaS tienen una alta escalabilidad vertical, se le permite al cliente completa flexibilidad en el uso de los sistemas operativos de su preferencia, o al cual pueda tener acceso.
- Las aplicaciones SaaS a menudo son personalizables y se pueden integrar con otras aplicaciones comerciales, especialmente entre aplicaciones de un proveedor de software común.

### 5.1.3. Inconvenientes

Estos son algunos de los inconvenientes de SaaS:

- La persona usuaria no tiene acceso directo a sus contenidos, ya que están guardados en un lugar remoto, salvo que el sistema prevea la exportación de los datos, y en caso de no contar con mecanismos de cifrado y control disminuye el índice de privacidad, control y seguridad que ello supone, ya que la compañía TI podría consultarlos.
- El usuario no tiene acceso al programa, por lo cual no puede hacer modificaciones (dependiendo de la modalidad del contrato de servicios que tenga con la compañía TI).
- Al estar el servicio y el programa dependientes de la misma empresa, no permite al usuario migrar a otro servicio utilizando el mismo programa (dependiendo de la modalidad del contrato de servicios con la compañía de TI).
- Si el servicio de Internet no está disponible por parte del Internet Service Provider (ISP), el usuario no tendrá acceso al programa, por lo que sus operaciones se verán afectadas hasta que dicho servicio se restablezca.
- Los clientes pierden el control sobre el control de versiones, si el proveedor adopta una nueva versión de una aplicación, la implementara para todos sus clientes, independiente de si el cliente quiere o no la versión más nueva. Esto puede requerir que la organización proporcione tiempo y recursos adicionales para la capacitación.
- La dificultad para cambiar de proveedor, ya que, los clientes deben de migrar grandes cantidades de datos.

### 5.1.4. Tipos

Algunos de los tipos de SaaS que existen son:

- Producto con autoservicio: se usa cuando el servicio crea bajo valor al cliente y en el mercado es fácil acceder a diferentes operadores. Esto causa bajos precios y mucha competitividad. Para usar este método de explotación es necesario que la aplicación sea fácil de comprar, de usar y libre de riesgos.
- Servicio personalizado de venta: se usa cuando el producto es algo complicado y puede requerir algún tipo de formación, el valor del cliente es variable, etc. En este sistema el cliente realiza una compra y la implementación necesita ser planeada y ejecutada adecuadamente. Además, normalmente el cliente quiere tener una relación personal con el proveedor y puede requerir cierto grado de calidad de servicio. Todos estos requisitos justifican precios más altos debido al coste extra generado.
- Producto para empresa: se usa en situaciones donde los beneficios del software como servicio (ej. bajo incremento de coste al crecer) ya no se logran, pero la flexibilidad y eficiencia de proveer el servicio, el mantenimiento y desarrollo permanece. Entonces se despliega la plataforma en la propia empresa. El proveedor casi siempre provee servicios de soporte, formación, consultoría, integración, etc., ya sea directamente o a través de un tercero.

## 5.2. DevOps

DevOps es una alineación de las operaciones de desarrollo y TI con una mejor comunicación y colaboración para realizar la producción e implementación de software de forma automatizada y repetible, lo cual ayuda a aumentar la velocidad de la organización a entregar servicios y aplicaciones de software (Taylor, [2022](#)).

### 5.2.1. Principios de DevOps

1. **Acción centrada en el cliente:** el equipo de DevOps debe tomar constantemente medidas centradas en el cliente para invertir en productos y servicios.
2. **Responsabilidad de extremo a extremo:** el equipo de DevOps debe proporcionar soporte de rendimiento del software hasta que llegue al final de su vida útil. Esto mejora el nivel de responsabilidad y la calidad de los productos diseñados.
3. **Mejora continua:** la cultura DevOps se centra en la mejora continua para minimizar el desperdicio y acelera continuamente la mejora de los productos o servicios ofrecidos.
4. **Automatice todo:** la automatización es un principio vital del proceso DevOps, y esto no es solo para el desarrollo de software, sino también para todo el panorama de la infraestructura.
5. **Trabaje como un solo equipo:** en la cultura DevOps, el diseñador, el desarrollador y el probador ya están definidos, y todo lo que necesitan hacer es trabajar como un solo equipo con una colaboración total.
6. **Supervise y pruebe todo:** el equipo de DevOps necesita procedimientos sólidos de supervisión y prueba.

### 5.2.2. Prácticas de DevOps

Las prácticas de DevOps son un reflejo de la idea de automatización y mejora continuas, y muchas de ellas se centran en una o en varias fases del ciclo de desarrollo (NetApp, [2019](#)). Estas prácticas incluyen lo siguiente:

- **Desarrollo continuo:** esta práctica abarca las fases de planificación y codificación del ciclo de DevOps. Puede incluir también mecanismos de control de versiones.
- **Realización de pruebas continuas:** esta práctica incorpora continuas pruebas de código automatizadas y programadas con antelación que se realizan a medida que el código de aplicación se está creando o actualizando. Gracias a estas pruebas, el código pasa antes a la fase de producción.
- **Integración continua (CI):** en esta práctica se combinan herramientas de gestión de configuración (CM) con otras herramientas de pruebas y desarrollo para saber qué cantidad del código que se está creando está listo para pasar a producción. Para ello, debe existir un intercambio fluido de información entre las fases de prueba y de desarrollo que permita identificar y resolver con rapidez problemas en el código.



- **Entrega continua:** esta práctica automatiza la introducción de cambios en el código para pasar a un entorno de preproducción o de almacenamiento provisional tras la fase de pruebas. Un miembro del equipo podría entonces decidir si es conveniente promover estos cambios de código a la fase de producción.
- **Puesta en marcha continua (CD):** al igual que sucede con la entrega continua, esta práctica automatiza el lanzamiento de código nuevo o modificado a la fase de producción. Una empresa que pone en práctica la puesta en marcha continua podría publicar cambios en código o funciones varias veces al día. Las tecnologías de contenedor, como Docker y Kubernetes, hacen posible esta fase de puesta en marcha continua al ayudar a mantener la coherencia del código entre los diferentes entornos y plataformas de puesta en marcha.
- **Supervisión continua:** esta práctica implica la supervisión continua del código en la fase de producción y la infraestructura subyacente que la sustenta. A través de un bucle de retroalimentación en el que se notifican errores o problemas, este podría volver a la fase de desarrollo.
- **Infraestructura como código:** esta práctica se puede utilizar durante varias fases de DevOps para automatizar el aprovisionamiento de la infraestructura que se necesita para publicar el software. Los desarrolladores añaden «código» de infraestructura procedente de las herramientas de desarrollo actuales. Por ejemplo, los desarrolladores podrían crear un volumen de almacenamiento bajo demanda desde Docker, Kubernetes u OpenShift. Gracias a esta práctica, los equipos de operaciones también pueden supervisar las configuraciones de entorno, registrar los cambios y simplificar la reversión de las configuraciones.

### 5.3. Control de versiones

El control de versiones, también conocido como "control de código fuente", es la práctica de rastrear y gestionar los cambios en el código de software. Los sistemas de control de versiones son herramientas de software que ayudan a los equipos de software a gestionar los cambios en el código fuente a lo largo del tiempo. A medida que los entornos de desarrollo se aceleran, los sistemas de control de versiones ayudan a los equipos de software a trabajar de forma más rápida e inteligente. Son especialmente útiles para los equipos de DevOps, ya que les ayudan a reducir el tiempo de desarrollo y a aumentar las implementaciones exitosas (Atlassian, 2022).

El software de control de versiones realiza un seguimiento de todas las modificaciones en el código en un tipo especial de base de datos. Si se comete un error, los desarrolladores pueden ir hacia atrás en el tiempo y comparar las versiones anteriores del código para ayudar a resolver el error, al tiempo que se minimizan las interrupciones para todos los miembros del equipo (Atlassian, 2022).

#### 5.3.1. Subversion

SVN se basa en un sistema de control de versiones centralizado. Esto significa que existe un almacén central de datos (el repositorio) accesible a todos los usuarios. Dado que los cambios realizados no pueden ser fusionados entre sí, el sistema evita que dos usuarios puedan editar un mismo

archivo al mismo tiempo. El proceso es muy simple, cuando uno de los usuarios accede a un archivo, el sistema lo marca automáticamente como de solo lectura para los demás. Además, Apache Subversion ofrece la posibilidad de descargar y editar directorios individuales sin depender del árbol general de directorios. De esta manera, es posible asignar diferentes permisos de lectura y escritura a los diferentes usuarios. Subversion se caracteriza también porque puede registrar directorios vacíos, renombrados y mudados de sitio sin pérdidas de su historia (IONOS, 2020).

### 5.3.2. Git

Git es un sistema de control de versiones distribuido, lo que significa que, aunque existe un repositorio central en el cual se incorporan los cambios, todos los usuarios pueden descargar su propia copia de trabajo. De esta forma, todos tienen acceso al repositorio completo, incluyendo el historial local, sin depender de ningún tipo de conexión de red. Todos los cambios se transfieren rápidamente al repositorio central. Como consecuencia, Git no ofrece ningún sistema de bloqueo, sino que cada usuario genera sus propios directorios o branches dentro del árbol para ser cargados posteriormente al repositorio central. Por defecto, cada usuario tiene permisos de lectura y escritura para los diferentes directorios (en caso de que se quiera asignar permisos especiales, será necesario crear otros directorios raíz). Cada copia de trabajo es una copia de seguridad independiente del directorio raíz, lo que resulta ventajoso si este sufre algún daño o fallo. Git solo registra los contenidos de los directorios, por eso los vacíos se eliminan automáticamente (IONOS, 2020).

## 5.4. Pruebas automatizadas

La práctica de automatización de pruebas consiste en revisar y validar automáticamente un producto de software (por ejemplo, una aplicación web) para asegurarse de que cumple con los estándares de calidad predefinidos para el estilo de código, la funcionalidad (lógica empresarial) y la experiencia del usuario (Hristov, 2021).

Las pruebas suelen constar de las siguientes etapas:

- **Pruebas unitarias:** verifican unidades individuales de código, como una función, para comprobar que funcionan según lo esperado.
- **Pruebas de integración:** garantizan que varios fragmentos de código puedan funcionar juntos sin consecuencias imprevistas.
- **Pruebas de extremo a extremo:** verifican que la aplicación cumple con las expectativas del usuario.
- **Pruebas exploratorias:** adoptan un enfoque no estructurado para revisar numerosas áreas de una aplicación desde la perspectiva del usuario con el fin de descubrir problemas funcionales o visuales.

Una práctica recomendada de DevOps es ejecutar pruebas automatizadas cuanto antes y con la mayor frecuencia posible dentro de las pipelines de CI/CD (Hristov, 2021).

## 5.5. Metodologías de CI/CD

CI/CD es un método para entregar aplicaciones con frecuencia a los clientes mediante la introducción de la automatización en las etapas de desarrollo de aplicaciones. CI/CD es una solución a los problemas que la integración de código nuevo puede causar a los equipos de desarrollo y operaciones. Específicamente, CI/CD introduce la automatización y el monitoreo continuos a lo largo del ciclo de vida de las aplicaciones, desde las fases de integración y prueba hasta la entrega y el despliegue. En conjunto, estas prácticas conectadas a menudo se denominan “pipeline de CI/CD” (Santiago Montoya, 2018).

### 5.5.1. ¿Cuál es la diferencia entre CI y CD en el concepto CI/CD?

El “CI” en CI/CD siempre se refiere a la integración continua, que es un proceso de automatización para los desarrolladores. El CI exitoso significa que los nuevos cambios de código en una aplicación se crean, prueban y fusionan regularmente en un repositorio compartido. Es una solución al problema de tener demasiadas ramas de una aplicación en desarrollo a la vez que pueden entrar en conflicto entre sí. El “CD” en CI/CD se refiere a la entrega continua y/o la implementación continua. La entrega continua generalmente significa que los cambios de un desarrollador en una aplicación se prueban automáticamente y se cargan en un repositorio, donde luego el equipo de operaciones puede implementarlos en un entorno de producción en vivo. El propósito de la entrega continua es garantizar que se requiera un esfuerzo mínimo para implementar código nuevo. La implementación continua puede referirse a la liberación automática de los cambios de un desarrollador del repositorio a producción, donde los clientes pueden utilizarlos. Aborda el problema de sobrecargar los equipos de operaciones con procesos manuales que ralentizan la entrega de aplicaciones (Santiago Montoya, 2018).

Figura 1: Diferencia entre las metodologías de CI/CD.



### 5.5.2. Beneficios de la implementación de CI/CD

- Cambios de código más pequeños: una ventaja técnica es que permite integrar pequeños fragmentos de código a la vez. Estos cambios de código son más simples y fáciles de manejar que grandes fragmentos de código y, como tales, tienen menos problemas que haya que solucionar más adelante.
- El tiempo medio de resolución es más rápido (MTTR): el MTTR mide la capacidad de mantenimiento de las funciones a reparar y establece el tiempo promedio para reparar una función errónea. Básicamente, ayuda a realizar un seguimiento de la cantidad de tiempo dedicado a

recuperarse de una falla. El CI / CD ayuda a reducir el MTTR porque los cambios de código son más pequeños y es más sencillo aislar los fallos.

- Tasa de liberación más rápida: al detectar los defectos más rápido y, de esta forma, repararlos más rápido, aumenta las tasas de liberación.
- Reducir costos: la automatización reduce la cantidad de errores que pueden tener lugar en los muchos pasos repetitivos. Hacerlo así también libera tiempo de los programadores que se van a dedicar a desarrollo de productos, ya que al detectar antes los errores se reducen las correcciones de código que se puedan producir en el futuro. Otra cosa que al aumentar la calidad del código con la automatización también aumenta su ROI.
- Mantenimiento y actualizaciones más sencillos: el mantenimiento y las actualizaciones son una parte crucial en el ciclo de vida de un producto. Sin embargo, es importante tener en cuenta que, dentro de un proceso, se debe realizar el mantenimiento durante la hora no crítica. Hacerlo en horas productivas puede afectar al cliente y aumentar los problemas de implementación por las prisas. Utilizando la CI/CD, se asegura de que los cambios en los sistemas productivos se realizan en momentos que no afecten a los clientes.

### 5.5.3. Desafíos

Algunos desafíos de la implementación de CI/CD son:

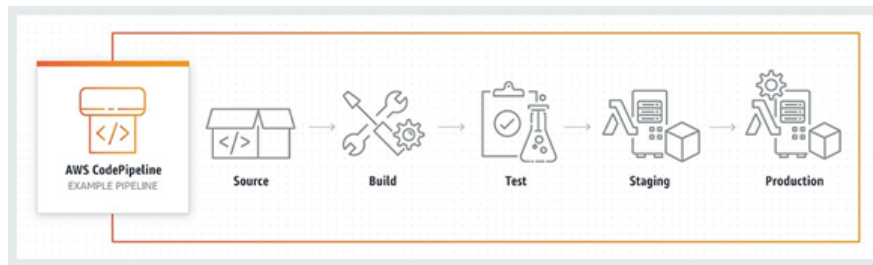
- La coherencia y la estandarización debido a que las aplicaciones y la infraestructura pueden no encajar en una caja unificada. Además, las plataformas no suelen tener el mismo tipo de código, lo que significa una duplicación de esfuerzos.
- La seguridad se convierte en parte de CI/CD, por lo que conseguir que la seguridad se adapte al proceso y no al revés puede ser un desafío.
- Existe una gama cada vez mayor de servicios basados en la nube para desarrollo, prueba, implementación, CI/CD y más. Con las herramientas de código abierto, cualquiera puede elegir el producto que mejor se adapte a sus habilidades y trabajo, por tanto, sin supervisión una organización puede terminar con docenas de herramientas innecesarias en uso.

### 5.5.4. Herramientas

Las herramientas más comunes para la implementación de CI/CD:

- AWS CodePipeline: es un servicio de entrega continua completamente administrado que permite automatizar pipelines de lanzamiento para lograr actualizaciones de infraestructura y aplicaciones rápidas y fiables. AWS CodePipeline usa otras herramientas como complemento para el desarrollo completo del proceso DevOps, como lo son AWS CodeCommit, AWS CodeBuild, AWS CodeDeploy y AWS CodeStar; todas ellas trabajan de manera integrada para asegurar un proceso de automatización completo (Santiago Montoya, 2020).

Figura 2: Funcionamiento de CI/CD en AWS CodePipeline.



- Azure pipelines: es un servicio en la nube que combina la integración y entrega continua para compilar y probar automáticamente código para finalmente ponerlo a disposición de los usuarios (Santiago Montoya, 2020).

Figura 3: Funcionamiento de Azure pipelines.

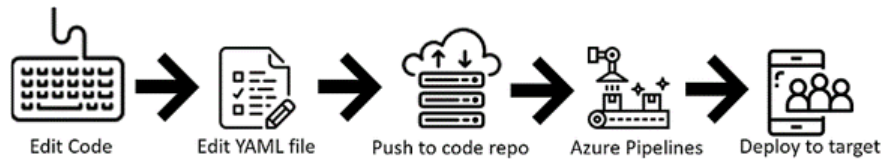
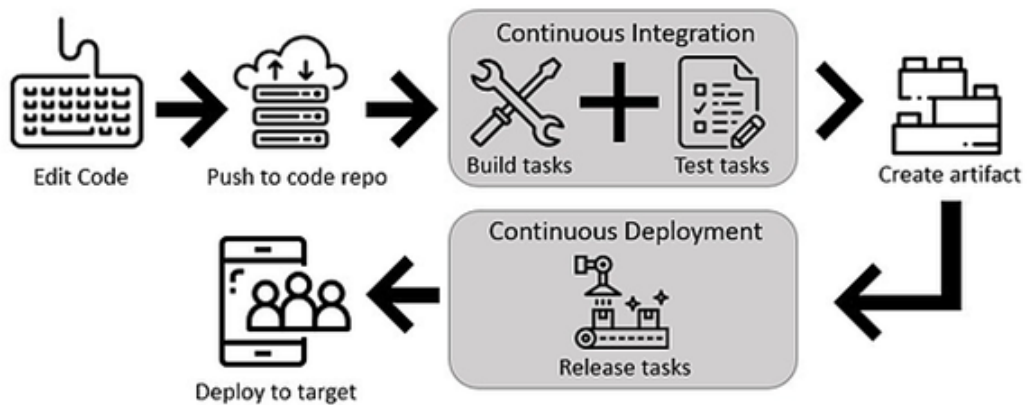
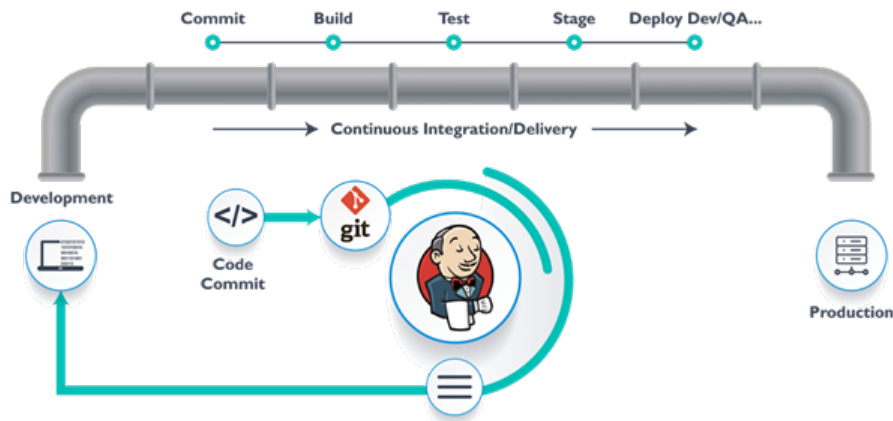


Figura 4: Funcionamiento de CI/CD con Azure pipelines.



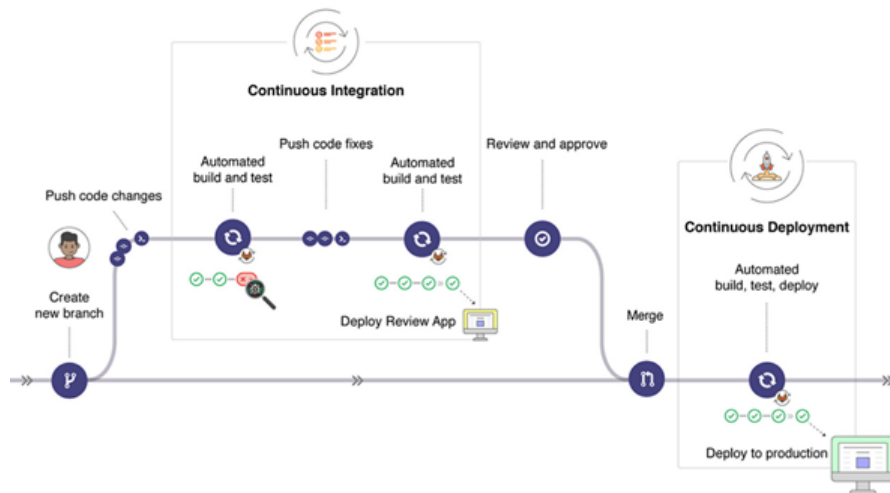
- Jenkins: es la herramienta de integración continua más conocida del mercado, está desarrollada en Java, es compatible con múltiples sistemas de control de versiones. Al ser de código abierto permite tenerlo de manera local simplemente cumpliendo ciertas especificaciones en las máquinas, y al mismo tiempo, poder configurarlo en entornos productivos (Santiago Montoya, 2020).

Figura 5: Funcionamiento de CI/CD con Jenkins.



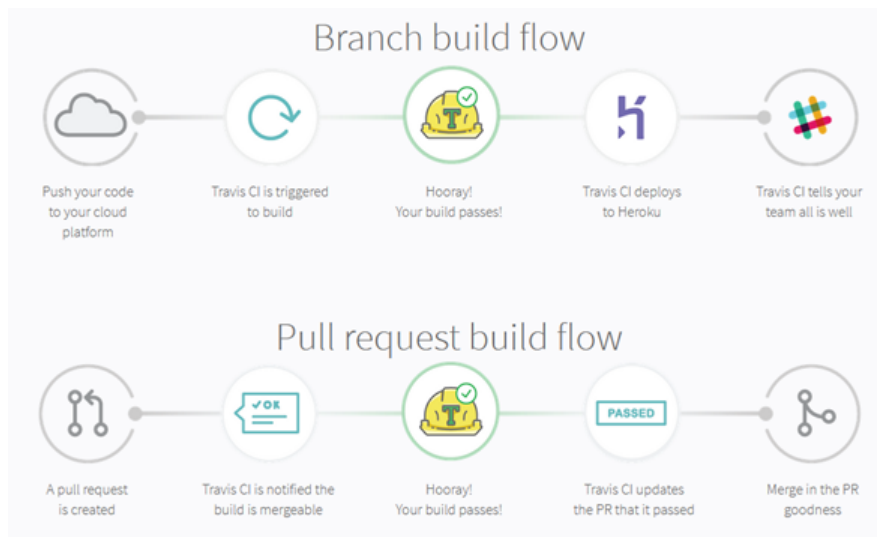
- GitLab CI/CD: es un servicio que viene incluido en GitLab, el cual construye y prueba el software cada vez que un desarrollador actualiza el repositorio. Dentro de la Suite de GitLab también existe GitLab CD, que se encarga de la entrega continua, después de que se completa la integración por medio de GitLab CI, se puede ejecutar la implementación continua para actualizar los cambios dentro del producto que ya está en producción (Santiago Montoya, 2020).

Figura 6: Funcionamiento de CI/CD con GitLab CI/CD.



- Travis CI: es una plataforma en la nube que funciona de manera gratuita para repositorios en GitHub públicos y BitBucket, esta herramienta ofrece opciones de integración continua automatizadas que eliminan la necesidad de un servidor dedicado, lo que permite realizar pruebas en diferentes entornos, en varias máquinas, y corriendo en diferentes sistemas operativos (Santiago Montoya, 2020).

Figura 7: Funcionamiento de CI/CD con Travis CI.



## 5.6. ¿Qué es un contenedor de Docker?

Un contenedor de Docker es un conocido contenedor ejecutable, independiente, ligero que integra todo lo necesario para ejecutar una aplicación, incluidas bibliotecas, herramientas del sistema, código y tiempo de ejecución. Docker es también una plataforma de software que permite a los desarrolladores crear, probar e implementar aplicaciones en contenedores de forma rápida (Oracle, 2021).

### 5.6.1. Terminología de Docker

- **Docker:** plataforma de contenedor de software diseñada para desarrollar, enviar y ejecutar aplicaciones aprovechando la tecnología de los contenedores.
- **Contenedor:** los contenedores comparten el núcleo del sistema host con otros contenedores. Un contenedor, que se ejecuta en el sistema operativo host, es una unidad de software estándar que empaqueta código y todas sus dependencias, para que las aplicaciones se puedan ejecutar de forma rápida y fiable de un entorno a otro. Los contenedores no son persistentes y se activan desde imágenes.
- **Motor de Docker:** el software de host de código abierto que crea y ejecuta los contenedores.
- **Imágenes de Docker:** colección de software que se ejecutará como un contenedor que incluye un conjunto de instrucciones para crear un contenedor que se pueda ejecutar en la plataforma Docker. Las imágenes no son modificables, de modo que para realizar cambios en una imagen es preciso crear otra nueva.
- **Docker Registry:** lugar para almacenar y descargar imágenes. Registry es una aplicación de servidor escalable y sin estado que almacena y distribuye imágenes de Docker.





### 6.1. Análisis de la situación actual

Para este proyecto, se obtuvieron los registros de tiempo de los últimos seis meses en la herramienta de manejo de tiempos Zoho Projects que le toma a un desarrollador crear instancias de la plataforma web y realizar una actualización.

Cuadro 1: Tiempos creación de instancias de usuario - instalación. Registros de tiempo utilizado por usuario para crear nuevas instalaciones.

A cargo de	Instalación	Tiempo (horas)
A	OE	1.58
A	I	0.93
D	SA	0.05
A	SA	2.75
H	PR	0.8
A	PR	1.97
A	NI	1.59
A	PP	0.59
A	AG	1.47
A	AT	1.15
A	AB	1.58
A	OM	0.83
C	SEL	1.32
A	AL	1.86
A	MG	1.45
A	ME	1.35

A	SN	2.66
---	----	------

Cuadro 2: Tiempos creación de instancias de instalaciones. Registros de tiempo utilizado para crear nuevas instalaciones.

Instalación	Tiempo (horas)
OE	1.58
I	0.93
SA	2.8
PR	2.77
NI	1.59
PP	0.98
AG	1.47
AT	1.15
AB	1.58
OM	0.83
SEL	1.32
AL	1.86
MG	1.45
ME	1.35
SN	2.66

Cuadro 3: Tiempos actualización de usuario - instalación. Registros de tiempo utilizado por usuario para actualizar una o múltiples instalaciones.

A cargo de	Instalación	Tiempo (horas)
A	GA	0.38
D	E	0.15
A	General	2.29
A	General	1.6
A	General	0.58
H	W	0.5
A	W	0.5
R	GR	0.15
A	General	3.68
A	General	6.12
A	General	2.12
A	W	1.65
C	General	1.6
A	General	0.5
A	General	3.57

Cuadro 4: Tiempos actualización de instalaciones. Registros de tiempo utilizado para actualizar una o múltiples instalaciones.

Instalación	Tiempo (horas)
GA	0.38
E	0.15
General	2.29
General	1.6
General	0.58
W	1
GR	0.15
General	3.68
General	6.12
General	2.12
W	1.65
General	1.6
General	0.5
General	3.57

## 6.2. Evaluación y realización de pruebas automatizadas

Se realizó la configuración necesaria en el proyecto para soportar una suite de pruebas automatizadas, realizando una prueba de concepto de las mismas. La herramienta seleccionada para el proceso es PHPUnit debido a que forma parte del grupo de frameworks de xUnit, tiene un conjunto completo de bibliotecas de JUnit para PHP, cuenta con soporte para Mock Objects (jMock) y tiene la posibilidad de integrarse con varias aplicaciones de test (PHPUnit, [2021](#)).

Instalar la librería de Composer para PHPUnit

```
composer require --dev phpunit/phpunit ^8.4
```

Realizar la configuración para utilizar PHPUnit con el framework de Symfony.

```
1 $_test_dir = realpath(dirname(__FILE__) . '/../');
2
3 require_once dirname(__FILE__) . '/../../config/ProjectConfiguration.
  class.php';
4 $configuration = ProjectConfiguration::hasActive() ?
  ProjectConfiguration::getActive() : new ProjectConfiguration(
  realpath($_test_dir . '/../'));
5
6 $autoload = sfSimpleAutoload::getInstance(sfConfig::get('sf_cache_dir')
  . '/project_autoload.cache');
```

```

7  $autoload->loadConfiguration(sfFinder::type('file')->name('autoload.yml'
   )->in(array(
8     sfConfig::get('sf_symfony_lib_dir') . '/config/config',
9     sfConfig::get('sf_config_dir'),
10  ));
11  $autoload->register();

```

Realizar la configuración para utilizar PHPUnit con el ORM de Propel.

```

1  include(dirname(__FILE__) . '/unit.php');
2
3  $configuration = ProjectConfiguration::getApplicationConfiguration('
   fatorh', 'test', true);
4
5  new sfDatabaseManager($configuration);
6
7  $loader = new sfPropelData();
8  $loader->loadData(sfConfig::get('sf_test_dir') . '/fixtures');

```

Crear los fixtures de test para los datos que serán utilizados durante las pruebas. Crear el archivo de pruebas y escribir las pruebas correspondientes. Por último, para ejecutar las pruebas unitarias se utiliza el comando

```
php lib/vendor/bin/phpunit CalculoPlanillaHelperTest.php
```

Figura 8: Ejecución de pruebas automatizadas.

```

nothing to commit, working tree clean
servir@devservir ~$ ./lib/vendor/bin/phpunit ./test/unit/CalculoPlanillaHelperTest.php
PHPUnit 8.5.30 #StandWithUkraine

..
                                         2 / 2 (100%)

Time: 8.33 seconds, Memory: 156.25 MB

OK (2 tests, 4 assertions)
servir@devservir ~$ ./lib/vendor/bin/phpunit ./test/unit/RhPlanillaResumenPeerTest.php
PHPUnit 8.5.30 #StandWithUkraine

.....
                                         9 / 9 (100%)

Time: 2.01 seconds, Memory: 160.25 MB

OK (9 tests, 17 assertions)
servir@devservir ~$

```

### 6.3. Deployer

Para la integración de nuevas funcionalidades al ambiente de producción se seleccionó la herramienta de Deployer por medio del uso de recetas debido a:

- Es una herramienta que fue lanzada en el año 2013, por lo cual tiene el respaldo necesario (Medvedev, 2013).
- Es Open-Source, por tanto, está disponible de forma gratuita.

- Tiene una comunidad activa encargada de darle soporte, obtiene parches y mejoras de forma constante.
- Es utilizado en cientos de miles de proyectos a nivel mundial, cada mes tiene más de un millón de implementaciones (Medvedev, 2022).
- Tiene soporte para más de 50 frameworks y servicios de terceros (Medvedev, 2013).

### 6.3.1. Implementación

Instalar en el proyecto la librería de Composer para Deployer

```
composer require --dev "deployer/deployer: ^6.0"
```

Inicializar Deployer en el proyecto

```
php lib/vendor/bin/dep init
```

Para la actualización de instancias ya existentes se crea el archivo base *base-deploy.php* donde se alojan las instrucciones generales para una actualización de la plataforma. Este extiende del archivo *recipe/common.php* proporcionado por Deployer. Se procede a agregar una serie de parámetros donde se define el nombre del proyecto, el repositorio de Gitlab y la cantidad de releases que se desean almacenar.

```
1 namespace Deployer;
2
3 require 'recipe/common.php';
4
5 set('application', 'factorh');
6
7 set('repository', 'git@gitlab.com:servir/factorh.git');
8
9 set('keep_releases', 2);
```

Definir el comportamiento esperado de los comandos de Deployer, según la necesidad del proyecto.

```
1 set('writable_mode', 'chmod');
2 set('writable_chmod_mode', '0777');
3 set('writable_use_sudo', true);
```

Identificar y definir los directorios y archivos compartidos entre releases.

```
1 set('shared_dirs', [
2     'log',
3     'web/uploads/aniversario',
4     'web/uploads/archivo_ss',
```

```

5     'web/uploads/archivos_csv',
6     'web/uploads/assets',
7     'web/uploads/banners',
8     'web/uploads/bonos_venta',
9     'web/uploads/cargas',
10    'web/uploads/cargas_subir',
11    'web/uploads/cargos_planilla',
12    'web/uploads/catalogo_dato/documentos',
13    'web/uploads/catalogo_dato/fotos',
14    'web/uploads/correos',
15    'web/uploads/cumpleaños',
16    'web/uploads/descriptor_puesto',
17    'web/uploads/descuentos',
18    'web/uploads/facturas',
19    'web/uploads/historico',
20    'web/uploads/ingresos',
21    'web/uploads/LibroSalarios',
22    'web/uploads/personal/firmas',
23    'web/uploads/personal/fotos',
24    'web/uploads/planificaciones',
25    'web/uploads/PlanificacionesV',
26    'web/uploads/PlanillaPorcentaje',
27    'web/uploads/vacaciones',
28    'web/uploads/modelo_referencia'
29 ];
30 set('shared_files', ['config/databases.yml', 'config/parameters.yml']);

```

Identificar y definir los directorios que pueden ser escritos por el servidor.

```

1 set('writable_dirs', ['cache', 'log', 'web/uploads']);

```

Si es necesario para el proyecto definir parámetros de configuración como funciones, para poder usarlos dentro de las instrucciones con `{{}}`.

```

1 set('console', function() {
2     return '{{release_path}}/symfony';
3 });

```

Crear tareas propias o personalizar las tareas que provee Deployer

```

1 after('deploy:failed', 'deploy:unlock');
2
3 task('deploy:plugin:publish-assets', function() {
4     run('{{bin/php}} {{console}} plugin:publish-assets');
5 });
6
7 task('deploy:cache:clear', function() {
8     run('{{bin/php}} {{console}} cc');
9     run('{{bin/php}} {{console}} cc --env=prod');
10

```

```

11     run('{{bin/php}} {{console}} cc --app=portalusuario');
12     run('{{bin/php}} {{console}} cc --env=prod --app=portalusuario');
13 });
14
15 task('database:migrate', function() {
16     run('{{bin/php}} {{console}} propel:diff');
17
18     $migrationsAvailable = run('ls {{release_path}}/lib/model/migration/
19         ');
20
21     if (!$migrationsAvailable) {
22         writeln('<info>No migrations required</info>');
23
24         return;
25     }
26
27     writeln('<info>Migrations available</info>');
28
29     run('{{bin/php}} {{console}} propel:migrate');
30     run('rm {{release_path}}/lib/model/migration/*');
31 })->desc('Migrate database');
32
33 task('database:migrate:failed', function() {
34     writeln('<error>Migration went wrong</error>');
35
36     run('rm {{release_path}}/lib/model/migration/*');
37 });
38
39 task('factorh:completar-tabla-menu', function() {
40     run('{{bin/php}} {{console}} factorh:completar-tabla-menu');
41
42     writeln('<info>Completada la sincronizacion del menu</info>');
43 });
44
45 task('deploy:writable-shared', function() {
46     $sharedPath = '{{deploy_path}}/shared';
47
48     $writableSharedDirs = ['log', 'web'];
49
50     foreach ($writableSharedDirs as $writableSharedDir) {
51         run('sudo chmod -R 0777 {{deploy_path}}/shared/' .
52             $writableSharedDir);
53     }
54 });
55
56 fail('database:migrate', 'database:migrate:failed');

```

Agrupar las tareas en grupos, en este caso la tarea que se va a ejecutar para poder actualizar la instalación.

```

1 task('deploy', [
2     'deploy:info',
3     'deploy:prepare',

```

```

4     'deploy:lock',
5     'deploy:release',
6     'deploy:update_code',
7     'deploy:clear_paths',
8     'deploy:shared',
9     'deploy:vendors',
10    'deploy:plugin:publish-assets',
11    'deploy:cache:clear',
12    'deploy:writable',
13    'deploy:writable-shared',
14    'deploy:symlink',
15    'factorh:completar-tabla-menu',
16    'database:migrate',
17    'deploy:unlock',
18    'cleanup',
19  ]->desc('Deploy your project');
20
21  after('deploy', 'success');

```

Para la creación de nuevas instancias se crea el archivo *build-deploy.php* donde se alojan las instrucciones generales para crear una instancia de la plataforma. Este extiende del archivo *base-deploy.php* para utilizar las configuraciones y tareas definidas previamente. También se agregan tareas propias del proceso de crear una nueva instancia de la plataforma. Empezando por la llamada al API de GoDaddy para crear el subdominio correspondiente.

```

1  namespace Deployer;
2
3  require 'config/base-deploy.php';
4
5  task('build:request-domain', function() {
6      $API_KEY = '';
7      $API_SECRET = '';
8      $dnsName = get('build_name');
9      $ipDomain = get('ip_domain');
10     $dnsRecords = "[{\"data\": \"\$ipDomain\", \"port\": 80, \"priority
11         \": 10, \"ttl\": 3600}]";
12     $url = "https://api.godaddy.com/v1/domains/factor-rh.com/records/A/
13         \$dnsName";
14
15     $header = [
16         'Authorization: sso-key ' . $API_KEY . ':' . $API_SECRET . ',
17         'Content-Type: application/json',
18         'Accept: application/json'
19     ];
20
21     $ch = curl_init();
22     $timeout = 60;
23
24     curl_setopt($ch, CURLOPT_URL, $url);
25     curl_setopt($ch, CURLOPT_FOLLOWLOCATION, true);
26     curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
27     curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, $timeout);

```



```

26  curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
27  curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'PUT');
28  curl_setopt($ch, CURLOPT_POSTFIELDS, $dnsRecords);
29  curl_setopt($ch, CURLOPT_HTTPHEADER, $header);
30  $result = curl_exec($ch);
31  curl_close($ch);
32  $result = json_decode($result, true);
33
34  if ($result) {
35      writeln('<error>Domain request went wrong</error>');
36  }
37  });

```

También fue necesario crear una tarea para conectarse al servidor de bases de datos, crear una nueva base de datos con el nombre que será definido en las variables. Crear un dump de la base de datos origen con las configuraciones mínimas para uso de la instalación y volcarla en la base de datos recién creado. Asimismo crear el archivo *databases.yml* con la información de la base de datos y del servidor en el cual se encuentra alojada.

```

1  task('build:config-db', function() {
2      writeln('Crear base de datos');
3
4      run('/usr/bin/mysql -h<<db-server>> -u<<user>> -p<<password>> -e "
5          DROP DATABASE IF EXISTS {{db_name}}"');
6      run('/usr/bin/mysql -h<<db-server>> -u<<user>> -p<<password>> -e "
7          CREATE DATABASE {{db_name}}"');
8      run('/usr/bin/mysqldump -h<<db-server>> -u<<user>> -p<<password>>
9          factorh_base --ignore-table=factorh_base.
10         general_bitacora_actualizacion --ignore-table=factorh_base.
11         bitacora_acceso | /usr/bin/mysql -h<<db-server>> -u<<user>> -p<<
12         password>> -D{{db_name}}');
13
14     run('cp {{release_path}}/config/databases.dist.yml {{deploy_path}}/
15         shared/config/databases.yml');
16     run('cp {{release_path}}/config/parameters.yml.dist {{deploy_path}}/
17         shared/config/parameters.yml');
18
19     run('/bin/sed -i "s/localhost/ubuntu-db-server.local/" {{deploy_path}}
20         /shared/config/databases.yml');
21     run('/bin/sed -i "s/factorh/{{db_name}}/" {{deploy_path}}/shared/
22         config/databases.yml');
23     run('/bin/sed -i "s/syuser/factorh/" {{deploy_path}}/shared/config/
24         databases.yml');
25     run('/bin/sed -i "s/sypass/ServirFactorh17!/" {{deploy_path}}/shared
26         /config/databases.yml');
27 });

```

Igualmente fue necesario crear una tarea para crear la configuración correspondiente en el servidor de Apache, para esperar que el dominio se encuentre disponible y generar el certificado HTTPS.

```

1  task('build:config-server', function() {

```

```

2  writeln('Realizar las configuraciones del servidor de Apache');
3
4  run('echo "<VirtualHost *:80>" | /usr/bin/sudo /usr/bin/tee /etc/
5  apache2/sites-available/{{build_name}}.factor-rh.com.conf');
6  run('echo "    ServerAlias {{build_name}}.factor-rh.com" | /usr/bin/
7  sudo /usr/bin/tee -a /etc/apache2/sites-available/{{build_name}}.
8  factor-rh.com.conf');
9  run('echo "" | /usr/bin/sudo /usr/bin/tee -a /etc/apache2/sites-
10 available/{{build_name}}.factor-rh.com.conf');
11 run('echo "#    Redirect "/" "https://{{build_name}}.factor-rh.com
12 /" | /usr/bin/sudo /usr/bin/tee -a /etc/apache2/sites-available
13 /{{build_name}}.factor-rh.com.conf');
14 run('echo "</VirtualHost>" | /usr/bin/sudo /usr/bin/tee -a /etc/
15 apache2/sites-available/{{build_name}}.factor-rh.com.conf');
16 run('echo "" | /usr/bin/sudo /usr/bin/tee -a /etc/apache2/sites-
17 available/{{build_name}}.factor-rh.com.conf');
18 run('echo "# <IfModule mod_ssl.c>" | /usr/bin/sudo /usr/bin/tee -a /
19 etc/apache2/sites-available/{{build_name}}.factor-rh.com.conf');
20 run('echo "#    <VirtualHost *:443>" | /usr/bin/sudo /usr/bin/tee -a
21 /etc/apache2/sites-available/{{build_name}}.factor-rh.com.conf')
22 ;
23 run('echo "#    DocumentRoot /srv/web-apps/{{build_name}}.factor
24 -rh.com/web" | /usr/bin/sudo /usr/bin/tee -a /etc/apache2/sites-
25 available/{{build_name}}.factor-rh.com.conf');
26 run('echo "#    ServerName {{build_name}}.factor-rh.com" | /usr/
27 bin/sudo /usr/bin/tee -a /etc/apache2/sites-available/{{
28 build_name}}.factor-rh.com.conf');
29 run('echo "" | /usr/bin/sudo /usr/bin/tee -a /etc/apache2/sites-
30 available/{{build_name}}.factor-rh.com.conf');
31 run('echo "#    SSLCertificateFile /etc/letsencrypt/live/{{
32 build_name}}.factor-rh.com/cert.pem" | /usr/bin/sudo /usr/bin/
33 tee -a /etc/apache2/sites-available/{{build_name}}.factor-rh.com
34 .conf');
35 run('echo "#    SSLCertificateKeyFile /etc/letsencrypt/live/{{
36 build_name}}.factor-rh.com/privkey.pem" | /usr/bin/sudo /usr/bin
37 /tee -a /etc/apache2/sites-available/{{build_name}}.factor-rh.
38 com.conf');
39 run('echo "#    Include /etc/letsencrypt/options-ssl-apache.conf
40 " | /usr/bin/sudo /usr/bin/tee -a /etc/apache2/sites-available
41 /{{build_name}}.factor-rh.com.conf');
42 run('echo "#    SSLCertificateChainFile /etc/letsencrypt/live/{{
43 build_name}}.factor-rh.com/chain.pem" | /usr/bin/sudo /usr/bin/
44 tee -a /etc/apache2/sites-available/{{build_name}}.factor-rh.com
45 .conf');
46 run('echo "" | /usr/bin/sudo /usr/bin/tee -a /etc/apache2/sites-
47 available/{{build_name}}.factor-rh.com.conf');
48 run('echo "#    <Directory /srv/web-apps/{{build_name}}.factor-rh
49 .com/web>" | /usr/bin/sudo /usr/bin/tee -a /etc/apache2/sites-
50 available/{{build_name}}.factor-rh.com.conf');
51 run('echo "#    AllowOverride all" | /usr/bin/sudo /usr/bin/
52 tee -a /etc/apache2/sites-available/{{build_name}}.factor-rh.com
53 .conf');
54 run('echo "#    Require all granted" | /usr/bin/sudo /usr/bin
55 /tee -a /etc/apache2/sites-available/{{build_name}}.factor-rh.

```

```

    com.conf');
23 run('echo "#           Allow from All" | /usr/bin/sudo /usr/bin/tee
    -a /etc/apache2/sites-available/{{build_name}}.factor-rh.com.
    conf');
24 run('echo "#           </Directory>" | /usr/bin/sudo /usr/bin/tee -a /
    etc/apache2/sites-available/{{build_name}}.factor-rh.com.conf');
25 run('echo "# | /usr/bin/sudo /usr/bin/tee -a /etc/apache2/sites-
    available/{{build_name}}.factor-rh.com.conf');
26 run('echo "#           Alias /sf /srv/web-apps/{{build_name}}.factor-rh.
    com/lib/vendor/symfony/symfony1/data/web/sf" | /usr/bin/sudo /
    usr/bin/tee -a /etc/apache2/sites-available/{{build_name}}.
    factor-rh.com.conf');
27 run('echo "#           <Directory \"/srv/web-apps/{{build_name}}.factor-
    rh.com/lib/vendor/symfony/symfony1/data/web/sf\">" | /usr/bin/
    sudo /usr/bin/tee -a /etc/apache2/sites-available/{{build_name
    }}.factor-rh.com.conf');
28 run('echo "#           AllowOverride All" | /usr/bin/sudo /usr/bin/
    tee -a /etc/apache2/sites-available/{{build_name}}.factor-rh.com
    .conf');
29 run('echo "#           Require all granted" | /usr/bin/sudo /usr/bin
    /tee -a /etc/apache2/sites-available/{{build_name}}.factor-rh.
    com.conf');
30 run('echo "#           Allow from All" | /usr/bin/sudo /usr/bin/tee
    -a /etc/apache2/sites-available/{{build_name}}.factor-rh.com.
    conf');
31 run('echo "#           </Directory>" | /usr/bin/sudo /usr/bin/tee -a /
    etc/apache2/sites-available/{{build_name}}.factor-rh.com.conf');
32 run('echo "#           </VirtualHost>" | /usr/bin/sudo /usr/bin/tee -a /etc
    /apache2/sites-available/{{build_name}}.factor-rh.com.conf');
33 run('echo "# </IfModule>" | /usr/bin/sudo /usr/bin/tee -a /etc/
    apache2/sites-available/{{build_name}}.factor-rh.com.conf');
34
35 run('/usr/bin/sudo a2ensite {{build_name}}.factor-rh.com.conf');
36 run('/usr/bin/sudo systemctl reload apache2');
37
38 $domain = get('build_name') . '.factor-rh.com';
39 $sleepTimeSeconds = 10;
40 $attempt = 0;
41 $maxAttempts = 30;
42 $success = false;
43 while ($attempt < $maxAttempts) {
44     try {
45         writeln(sprintf('Making %s attempt to ping new domain',
46             $attempt));
47         $result = run("ping -c 1 $domain");
48
49         $success = true;
50         break;
51     } catch (\Deployer\Exception\RuntimeException $e) {
52         sleep($sleepTimeSeconds);
53         $attempt += 1;
54     }
55 }

```

```

56     if (!$success) {
57         writeln('<error>Error during certificate generation</error>');
58
59         return;
60     }
61
62     run('/usr/bin/sudo certbot --redirect -d {{build_name}}.factor-rh.
        com');
63     run('/usr/bin/sudo rm /etc/apache2/sites-available/{{build_name}}.
        factor-rh.com-le-ssl.conf');
64
65     run('echo "<VirtualHost *:80>" | /usr/bin/sudo /usr/bin/tee /etc/
        apache2/sites-available/{{build_name}}.factor-rh.com.conf');
66     run('echo "    ServerAlias {{build_name}}.factor-rh.com" | /usr/bin/
        sudo /usr/bin/tee -a /etc/apache2/sites-available/{{build_name}}.
        factor-rh.com.conf');
67     run('echo "" | /usr/bin/sudo /usr/bin/tee -a /etc/apache2/sites-
        available/{{build_name}}.factor-rh.com.conf');
68     run('echo "    Redirect "/" "https://{{build_name}}.factor-rh.com/"
        | /usr/bin/sudo /usr/bin/tee -a /etc/apache2/sites-available/{{
        build_name}}.factor-rh.com.conf');
69     run('echo "</VirtualHost>" | /usr/bin/sudo /usr/bin/tee -a /etc/
        apache2/sites-available/{{build_name}}.factor-rh.com.conf');
70     run('echo "" | /usr/bin/sudo /usr/bin/tee -a /etc/apache2/sites-
        available/{{build_name}}.factor-rh.com.conf');
71     run('echo "<IfModule mod_ssl.c>" | /usr/bin/sudo /usr/bin/tee -a /
        etc/apache2/sites-available/{{build_name}}.factor-rh.com.conf');
72     run('echo "    <VirtualHost *:443>" | /usr/bin/sudo /usr/bin/tee -a /
        etc/apache2/sites-available/{{build_name}}.factor-rh.com.conf');
73     run('echo "        DocumentRoot /srv/web-apps/{{build_name}}.factor-
        rh.com/web" | /usr/bin/sudo /usr/bin/tee -a /etc/apache2/sites-
        available/{{build_name}}.factor-rh.com.conf');
74     run('echo "        ServerName {{build_name}}.factor-rh.com" | /usr/
        bin/sudo /usr/bin/tee -a /etc/apache2/sites-available/{{
        build_name}}.factor-rh.com.conf');
75     run('echo "" | /usr/bin/sudo /usr/bin/tee -a /etc/apache2/sites-
        available/{{build_name}}.factor-rh.com.conf');
76     run('echo "        SSLCertificateFile /etc/letsencrypt/live/{{
        build_name}}.factor-rh.com/cert.pem" | /usr/bin/sudo /usr/bin/
        tee -a /etc/apache2/sites-available/{{build_name}}.factor-rh.com
        .conf');
77     run('echo "        SSLCertificateKeyFile /etc/letsencrypt/live/{{
        build_name}}.factor-rh.com/privkey.pem" | /usr/bin/sudo /usr/bin
        /tee -a /etc/apache2/sites-available/{{build_name}}.factor-rh.
        com.conf');
78     run('echo "        Include /etc/letsencrypt/options-ssl-apache.conf"
        | /usr/bin/sudo /usr/bin/tee -a /etc/apache2/sites-available/{{
        build_name}}.factor-rh.com.conf');
79     run('echo "        SSLCertificateChainFile /etc/letsencrypt/live/{{
        build_name}}.factor-rh.com/chain.pem" | /usr/bin/sudo /usr/bin/
        tee -a /etc/apache2/sites-available/{{build_name}}.factor-rh.com
        .conf');
80     run('echo "" | /usr/bin/sudo /usr/bin/tee -a /etc/apache2/sites-
        available/{{build_name}}.factor-rh.com.conf');

```

```

81 run('echo "      <Directory /srv/web-apps/{{build_name}}.factor-rh.
      com/web>" | /usr/bin/sudo /usr/bin/tee -a /etc/apache2/sites-
      available/{{build_name}}.factor-rh.com.conf');
82 run('echo "      AllowOverride all" | /usr/bin/sudo /usr/bin/tee
      -a /etc/apache2/sites-available/{{build_name}}.factor-rh.com.
      conf');
83 run('echo "      Require all granted" | /usr/bin/sudo /usr/bin/
      tee -a /etc/apache2/sites-available/{{build_name}}.factor-rh.com
      .conf');
84 run('echo "      Allow from All" | /usr/bin/sudo /usr/bin/tee -a
      /etc/apache2/sites-available/{{build_name}}.factor-rh.com.conf'
      );
85 run('echo "      </Directory>" | /usr/bin/sudo /usr/bin/tee -a /etc/
      apache2/sites-available/{{build_name}}.factor-rh.com.conf');
86 run('echo "" | /usr/bin/sudo /usr/bin/tee -a /etc/apache2/sites-
      available/{{build_name}}.factor-rh.com.conf');
87 run('echo "      Alias /sf /srv/web-apps/{{build_name}}.factor-rh.com
      /lib/vendor/symfony/symfony1/data/web/sf" | /usr/bin/sudo /usr/
      bin/tee -a /etc/apache2/sites-available/{{build_name}}.factor-rh
      .com.conf');
88 run('echo "      <Directory \"/srv/web-apps/{{build_name}}.factor-rh.
      com/lib/vendor/symfony/symfony1/data/web/sf\">" | /usr/bin/sudo
      /usr/bin/tee -a /etc/apache2/sites-available/{{build_name}}.
      factor-rh.com.conf');
89 run('echo "      AllowOverride All" | /usr/bin/sudo /usr/bin/tee
      -a /etc/apache2/sites-available/{{build_name}}.factor-rh.com.
      conf');
90 run('echo "      Require all granted" | /usr/bin/sudo /usr/bin/
      tee -a /etc/apache2/sites-available/{{build_name}}.factor-rh.com
      .conf');
91 run('echo "      Allow from All" | /usr/bin/sudo /usr/bin/tee -a
      /etc/apache2/sites-available/{{build_name}}.factor-rh.com.conf'
      );
92 run('echo "      </Directory>" | /usr/bin/sudo /usr/bin/tee -a /etc/
      apache2/sites-available/{{build_name}}.factor-rh.com.conf');
93 run('echo "      </VirtualHost>" | /usr/bin/sudo /usr/bin/tee -a /etc/
      apache2/sites-available/{{build_name}}.factor-rh.com.conf');
94 run('echo "</IfModule>" | /usr/bin/sudo /usr/bin/tee -a /etc/apache2
      /sites-available/{{build_name}}.factor-rh.com.conf');
95
96 run('/usr/bin/sudo systemctl reload apache2');
97 run('/bin/ln -s {{deploy_path}}/current /srv/web-apps/{{build_name
      }}.factor-rh.com');
98 });

```

**Agrupar las tareas en la tarea que se va a ejecutar para poder crear una nueva instancia.**

```

1 task('build', [
2     'build:request-domain',
3     'deploy:info',
4     'deploy:prepare',
5     'deploy:release',
6     'deploy:update_code',

```

```

7     'deploy:clear_paths',
8     'deploy:shared',
9     'build:config-db',
10    'deploy:vendors',
11    'deploy:plugin:publish-assets',
12    'deploy:cache:clear',
13    'deploy:writable',
14    'deploy:writable-shared',
15    'deploy:symlink',
16    'database:migrate',
17    'factorh:completar-tabla-menu',
18    'build:config-server',
19    'cleanup',
20 ]->desc('Build your project');
21
22 after('build', 'success');

```

En la branch específica del cliente definimos las variables correspondientes a esa instancia específica, las cuales serán utilizadas en la creación y actualización del recurso.

```

1 namespace Deployer;
2
3 require 'config/build-deploy.php';
4
5 set('branch', '<<branch>>');
6 host('<<host>>')
7     ->user('<<user>>')
8     ->set('build_name', '<<name>>')
9     ->set('db_name', '<<db_name>>')
10    ->set('ip_domain', '<<ip>>')
11    ->set('deploy_path', '<<path>>');

```

## 6.4. Gitlab CI/CD

La herramienta seleccionada es capaz de manejar eficientemente proyectos pequeños a grandes. GitLab incluye herramientas que facilitan el desarrollo de software al comprobar constantemente el código y asegurarse de que todo va bien (Encora, [2020](#)).

La integración continua (CI) funciona empujando el código a una aplicación alojada en un repositorio Git. En cada push, la aplicación ejecuta un canal de scripts para construir, probar y validar los cambios de código antes de fusionarlos con la branch principal. La implementación continua (CD) va un paso más allá que la CI, desplegando la aplicación en un servidor de producción en cada push, en la branch por defecto del repositorio (Encora, [2020](#)).

GitLab está construido con su propio y potente sistema de integración continua e implementación continua. Para configurar la CI/CD para un proyecto de GitLab, se debe colocar un archivo de configuración llamado *gitlab-ci.yml* en la raíz del repositorio. El archivo será ejecutado por el GitLab Runner, donde se realizará la construcción y prueba de la aplicación en cada push al reposi-

torio (Encora, 2020).

El GitLab Runner es un proyecto de código abierto que se utiliza para ejecutar trabajos y enviar los resultados a GitLab. Este servicio se incluye con GitLab y se utiliza junto con él para ejecutar el script del canal desde la CI y la CD (Encora, 2020).

- Es una herramienta gratuita que proporciona administración de repositorios de git, seguimiento de problemas, revisiones de código, wikis y fuente de actividades (Jain, 2020).
- Permite conexión con Active Directory y servidores LDAP para autorización y autenticación seguras (Jain, 2020).
- Es una herramienta integral de DevOps. Además proporciona planificación, empaquetado, SCM, lanzamiento, configuración y escrutinio (Jain, 2020).
- Dado que proporciona repositorios, su utilización es bastante sencilla y directa (Jain, 2020).
- Tiene una sólida documentación, un fácil control y una buena experiencia de usuario (Jain, 2020).
- Tiene una calificación de 4.5 de 5 estrellas con 635 reseñas en G2 (G2, 2022).
- Tiene una calificación de 8.6 de 10 puntos posibles en la categoría de fácil de usar en G2 (G2, 2022).
- Tiene una calificación de 8.5 de 10 puntos posibles en la categoría de calidad de soporte en G2 (G2, 2022).
- Tiene una calificación de 8.4 de 10 puntos posibles en la categoría de fácil de configurar en G2 (G2, 2022).
- Acorde a G2 está enfocado a la industria del software, servicios y tecnologías de la información. Teniendo un 38 % del mercado de las pequeñas empresas y un 36 % de las medianas empresas (G2, 2022).

#### 6.4.1. Migrar de SVN a GitLab

En este caso la plataforma se encontraba versionaba en SVN, se tomó la decisión de realizar la migración a Git que es el sistema de control de versiones en el que se fundamente GitLab, y así poder aprovechar todas las herramientas de GitLab y centralizar las funciones en un lugar.

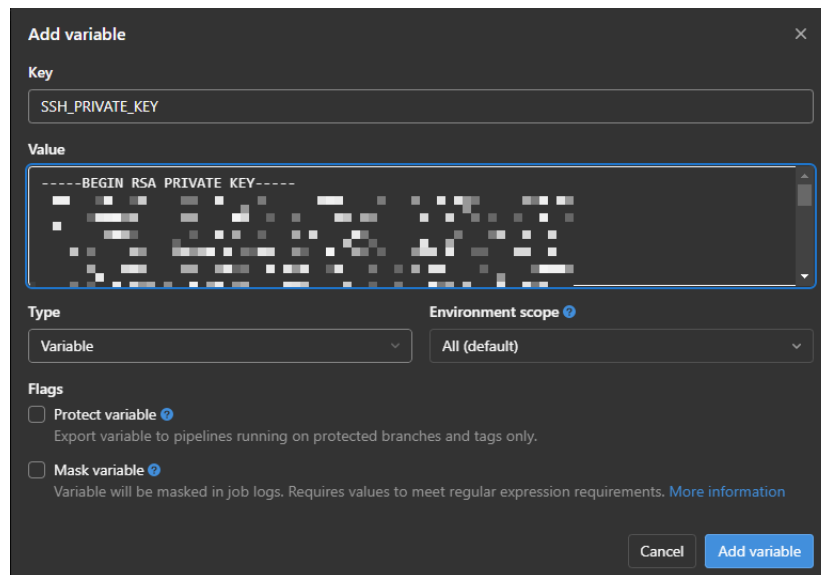
Para realizar la migración de SVN a GitLab fue utilizado *svn2git*. Fue necesario instalar *svn2git*, preparar el archivo de autores para asociar los commits y autores de SVN con los de GitLab.

Se creó el proyecto en GitLab para migrar el código, procesar el historial del proyecto en SVN y pushearlos hacia el nuevo repositorio de GitLab.



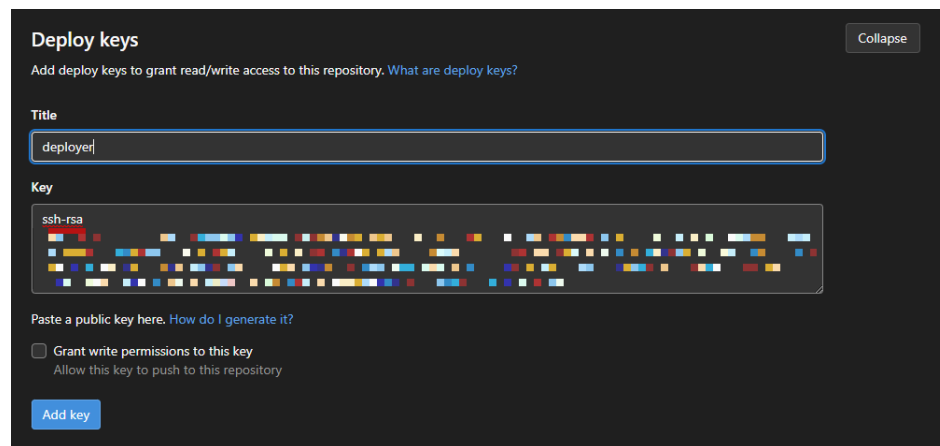


Figura 11: Agregar clave privada a variables de CI/CD.



También agregamos la clave pública **Project > Settings > Repository** como clave de implementación para poder acceder al repositorio a través de SSH.

Figura 12: Agregar clave pública como clave de implementación.



### 6.4.3. Crear una imagen de contenedor

GitLab CI/CD permite utilizar el motor de Docker para prueba e implementación de la plataforma. Se creó una imagen de Docker que tiene los requisitos mínimos que necesita una aplicación de Symfony para ejecutarse. Utilizar Docker es de las formas más eficientes de ejecutar las compilaciones. Fue creado un archivo Dockerfile en el directorio de la aplicación, el cual consiste en una instalación mínima para PHP.

```

1 FROM php:7.2-apache
2 RUN apt-get update
3 RUN apt-get install -qq git curl libmcrypt-dev libjpeg-dev libpng-dev
  libfreetype6-dev libbz2-dev libssl-dev libicu-dev libxml2-dev
4
5 RUN apt-get clean
6 RUN echo 'memory_limit = 2048M' >> /usr/local/etc/php/conf.d/docker-php-
  memlimit.ini;
7 RUN docker-php-ext-install gd iconv intl mbstring pdo phar xml zip
  pdo_mysql
8
9 RUN curl --silent --show-error "https://getcomposer.org/installer" | php
  -- --install-dir=/usr/local/bin --filename=composer

```

#### 6.4.4. Configuración del registro de contenedores de GitLab

Como siguiente paso, se compiló el archivo Dockerfile y se agragó al registro de contenedores de GitLab. Este registro permite almacenar y etiquetar imágenes para su uso.

Figura 13: Compilación de archivo Dockerfile.

```

C:\Windows\System32\cmd.exe
#8 1.666 bcmath bz2 calendar ctype curl dba dom enchant exif fileinfo filter ftp gd gettext gmp hash iconv imap interbas
e intl json ldap mbstring mysqli oci8 odbc opcache pcntl pdo pdo_dblib pdo_firebird pdo_mysql pdo_oci pdo_odbc pdo_pgsql
pdo_sqlite pgsqll phar posix pspell readline recode reflection session shmop simplexml snmp soap sockets sodium spl stan
dard sysvmsg sysvsem sysvshm tidy tokenizer wddx xml xmlreader xmlrpc xmlwriter xsl zend_test zip
#8 1.666
#8 1.666 Some of the above modules are already compiled into PHP; please check
#8 1.666 the output of "php -i" to see which modules are already loaded.
-----
executor failed running [/bin/sh -c docker-php-ext-install mcrypt pdo_mysql zip]: exit code: 1

[+] Building 30.2s (10/10) FINISHED
-> [internal] load build definition from Dockerfile 0.0s
-> => transferring dockerfile: 686B 0.0s
-> [internal] load .dockerignore 0.0s
-> => transferring context: 2B 0.0s
-> [internal] load metadata for docker.io/library/php:7.2 0.5s
-> [1/6] FROM docker.io/library/php:7.2@sha256:42ffbc0798e4449bbd1e14fc4dcb87774aa1ad1900a09ef6a965bc0880aa2161 0.0s
-> CACHED [2/6] RUN apt-get update 0.0s
-> CACHED [3/6] RUN apt-get install -qq git curl libmcrypt-dev libjpeg-dev libpng-dev libfreetype6-dev libbz2-de 0.0s
-> CACHED [4/6] RUN apt-get clean 0.0s
-> [5/6] RUN docker-php-ext-install pdo_mysql zip 26.0s
-> [6/6] RUN curl --silent --show-error "https://getcomposer.org/installer" | php -- --install-dir=/usr/local/bi 3.0s
-> exporting to image 0.5s
-> => exporting layers 0.4s
-> => writing image sha256:71b5462e49f6dba32c3bcdadced9b481bc15cbdae44dc780249edca66dcded 0.0s
-> => naming to registry.gitlab.com/servir/factorh 0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
D:\Servir\GENERAL\factorh>

```

Fue necesario instalar Docker en un equipo local de desarrollo e iniciar sesión con mi usuario de GitLab, para luego construir y enviar la imagen.

Figura 14: Agregar imagen de Docker al registro.

```
D:\Servir\GENERAL\factorh>docker push registry.gitlab.com/servir/factorh
Using default tag: latest
The push refers to repository [registry.gitlab.com/servir/factorh]
662eeda325a5: Pushed
302efe754baa: Pushed
4502a78783b2: Pushed
1532a401f18f: Pushed
313f1b7194ff: Pushed
95dc611e4b93: Pushed
01843c33a097: Pushed
b08f37a651df: Pushed
f677f1bb00d8: Pushed
b7f8cd6f6c5a: Pushed
df0a3def5414: Pushed
02eef72b445f: Pushed
e45a78df7536: Pushed
ddcd8d2fcf7e: Pushed
87c8a1d8f54f: Pushed
latest: digest: sha256:b2597512c2e40df2bc0b69459b599e8a67a0881218c84bd3a8a55a04faf90a7a size: 3458
D:\Servir\GENERAL\factorh>
```

### 6.4.5. Configuración de Gitlab CI/CD

Para compilar y utilizar la plataforma con Gitlab CI/CD, se necesita un archivo `.gitlab-ci.yml` en la raíz de nuestro repositorio.

Los runners ejecutan las instrucciones definidas en el `.gitlab-ci.yml`, la palabra `image` les indica a los runners que imagen utilizar. La palabra `services` indica las imágenes adicionales vinculadas a la imagen adicional. Se utiliza la imagen creada previamente como imagen principal y MySQL 5.7 como un servicio.

```
1 image: registry.gitlab.com/servir/factorh
2
3 services:
4   - mysql:5.7
```

Gitlab CI/CD permite el uso de variables, ya que definimos MySQL como administrador de bases de datos, el cual tiene un superusuario definido por defecto. Debemos ajustar la configuración de la instancia de MySQL definiendo las variables de `MYSQL_DATABASE` como el nombre de la base de datos y `MYSQ_ROOT_PASSWORD` como la contraseña de `root`. También definimos la URL de la plataforma y la branch que estaremos utilizando.

```
1 variables:
2   MYSQL_DATABASE: <<db_test>>
3   MYSQL_ROOT_PASSWORD: <<db_password>>
4
5   BASE_URL: <<URL>>
6   ACTIVE_BRANCH: <<branch>>
```

Definimos cada una de las etapas que se van a ejecutar en el pipeline.

```
1 stages:
2   - test
3   - deploy
```

En el proyecto agregamos el `config/parameters.gitlab-ci.yml` y el `config/databases.gitlab-ci.yml` con la configuración de la base de datos de pruebas, para poderlo usar en esa fase del pipeline.

Definimos como `.prepare_test` los script de shell se definen con la palabra clave `before_script`, los cuales se estarán ejecutando antes de las etapas que se definen posteriormente, estos son algunos comandos para preparar el ambiente de la plataforma y ejecutar las pruebas.

```
1 .prepare_test:
2   before_script:
3     - composer install --prefer-dist --optimize-autoloader
4     - cp config/parameters.gitlab-ci.yml config/parameters.mysql
5     - cp config/databases.gitlab-ci.yml config/databases.yml
6     - php symfony propel:diff
7     - php symfony propel:migrate
```

Definimos las tareas a ejecutarse de forma paralela, incluimos el `.prepare_test` para que lo utilicen y los script de shell se definen con la palabra clave `script`, con el comando para ejecutar las pruebas.

```
1 unit_test_rhplanillaresumenpeer:
2   stage: test
3   extends: .prepare_test
4   script:
5     - ./lib/vendor/bin/phpunit ./test/unit/RhPlanillaResumenPeerTest.php
6
7 unit_test_calculoplanillahelper:
8   stage: test
9   extends: .prepare_test
10  script:
11    - ./lib/vendor/bin/phpunit ./test/unit/CalculoPlanillaHelperTest.php
```

Para implementar en el servidor de producción con Deployer fue necesario configurar `SSH_PRIVATE_KEY` como una clave privada de SSH. La palabra clave `environment` le indica a GitLab que esta tarea se implementa en el entorno de producción. La palabra clave `url` se utiliza para generar un enlace a nuestra aplicación desde la página de entornos de GitLab.

```
1 .install_dependencies:
2   before_script:
3     - 'which ssh-agent || ( apt-get update -y && apt-get install openssh-client
4       -y )'
5     - eval $(ssh-agent -s)
6     - ssh-add <(echo "$SSH_PRIVATE_KEY")
7     - mkdir -p ~/.ssh
8     - '[[ -f /.dockerenv ]] && echo -e "Host *\n\tStrictHostKeyChecking no\n\n"
9       > ~/.ssh/config'
10    - composer install --prefer-dist --optimize-autoloader
11
12 deploy_production:
13   stage: deploy
14   extends: .install_dependencies
15   script:
16     - ./lib/vendor/bin/dep deploy
17   environment:
```

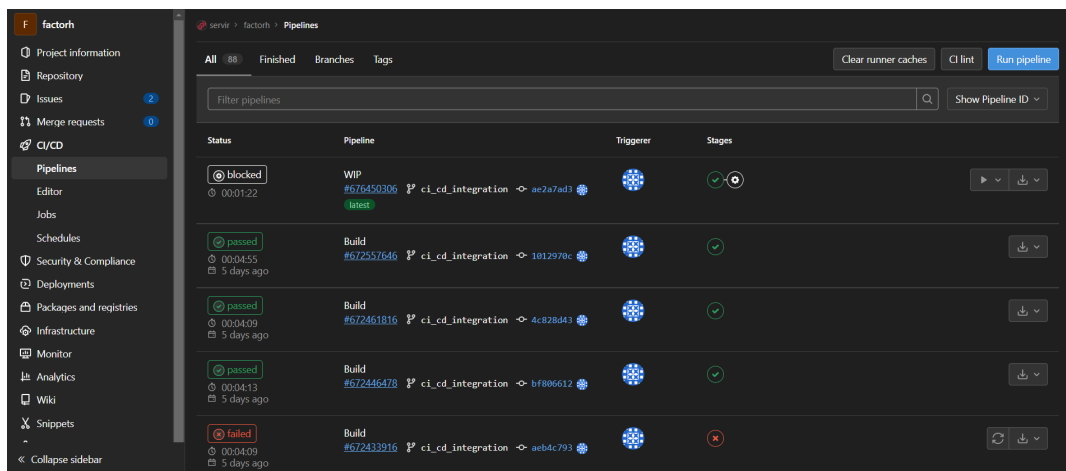
```

16   name: production
17   url: "$BASE_URL"
18   rules:
19     - if: $CI_COMMIT_TITLE == "Merge de origin/master" && $CI_COMMIT_BRANCH ==
      $ACTIVE_BRANCH
20     when: always
21     - if: $CI_COMMIT_TITLE != "Merge de origin/master" && $CI_COMMIT_BRANCH ==
      $ACTIVE_BRANCH
22     when: manual
23
24   build_production:
25     stage: deploy
26     extends: .install_dependencies
27     script:
28       - ./lib/vendor/bin/dep build
29     environment:
30       name: production
31       url: "$BASE_URL"
32     rules:
33     - if: $CI_COMMIT_TITLE == "Build" && $CI_COMMIT_BRANCH == $ACTIVE_BRANCH
34     when: always

```

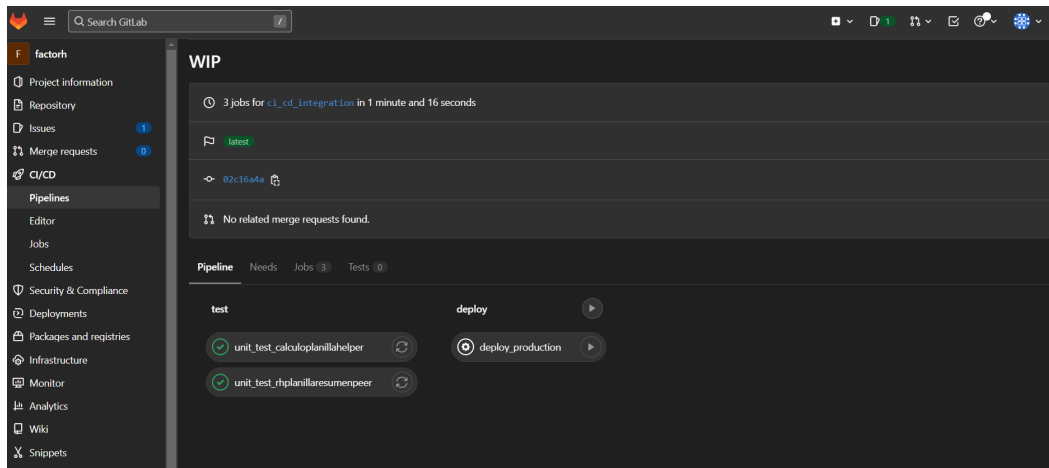
Al pushear el `.gitlab-ci.yml` al repositorio se activa nuestra pipeline que se puede visualizar en la sección de **Pipelines**.

Figura 15: Pipelines del proyecto



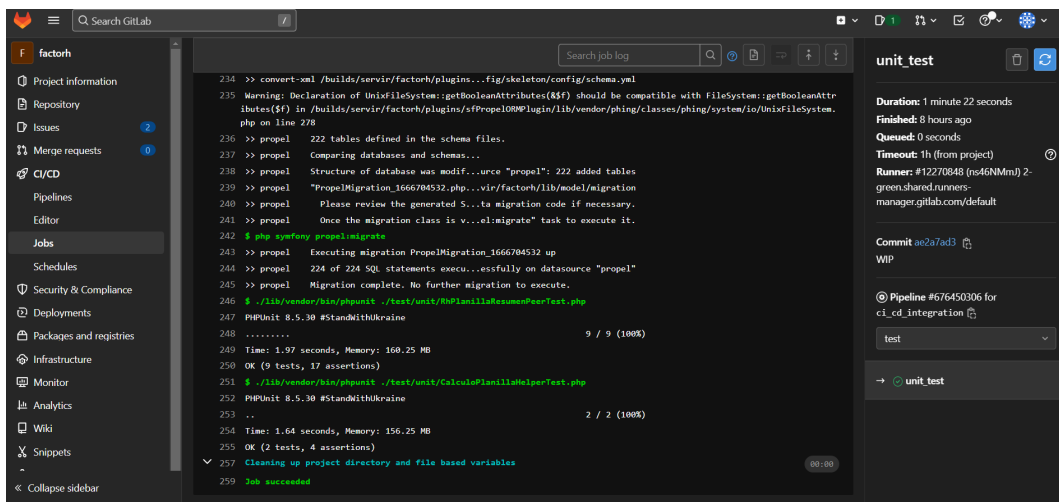
Ahí es posible visualizar las etapas de prueba e implementación.

Figura 16: Etapas del pipeline



Al seleccionar una de las etapas podemos visualizar la salida de los runners.

Figura 17: Etapas del pipeline



También en el servidor de producción es posible visualizar la estructura de la plataforma con Deployer, en el cual tenemos los tres directorios principales *current*, *releases* y *shared*. En el directorio de *current* realmente es un enlace simbólico que apunta a la última versión, y *shared* contiene los archivos que deben ser compartidos entre releases.

Figura 18: Estructura de la plataforma con Deployer

```
administrator@ubuntu-cloud-server:/mnt/ubuntu_cloud_server_external/web-apps$ tree -L 2 prueba-ci-cd-5.factor-rh.com/
prueba-ci-cd-5.factor-rh.com/
├── current -> releases/1
├── releases
│   └── 1
├── shared
│   ├── config
│   ├── ...
│   └── ...
7 directories, 8 files
administrator@ubuntu-cloud-server:/mnt/ubuntu_cloud_server_external/web-apps$
```

## 6.5. Integración con plataformas de notificación y manejo de errores

### 6.5.1. GitLab + Sentry

GitLab permite integraciones con Sentry para el seguimiento de versiones y acciones sugeridas para nuevos errores. Es posible realizar la conexión entre los errores de Sentry y los Issues de GitLab para la clasificación, asignación, reproducción, seguimiento y solución de errores (Sentry, 2022).

### 6.5.2. GitLab + Slack

La integración de notificaciones permite que el proyecto de GitLab envíe eventos, como la creación de problemas, a los equipos de Slack como notificaciones (GitLab, 2022).

### 6.5.3. Deployer + Sentry + Slack

A partir de la versión de Deployer lanzada el 10 de septiembre del 2022, Deployer permite integraciones con plataformas como Sentry y Slack para notificar acerca de un proceso completado exitosamente o bien de errores o rollbacks ejecutados (Medvedev, 2022).





La migración del código fuente de la herramienta de Subversion a Git, permitió conservar el historial completo de commits realizados.

### 7.1. Pipelines

Las pipelines de prueba en ambiente de producción de la plataforma web concluyeron exitosamente.

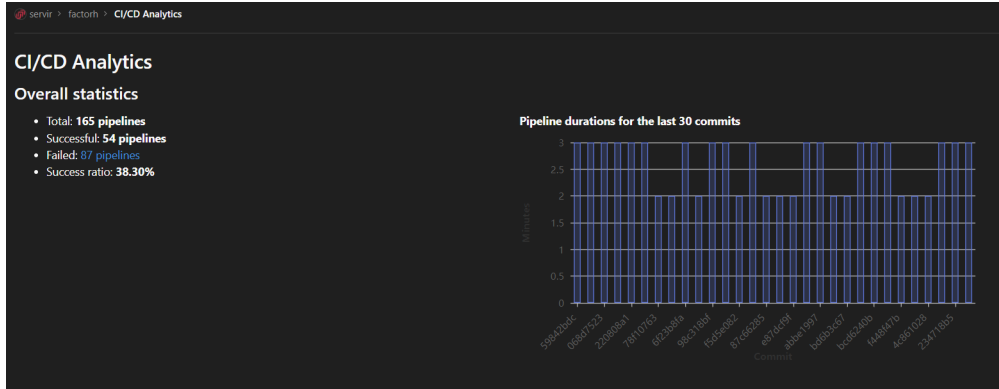
Figura 19: Pipelines realizadas exitosamente.

Status	Pipeline	Triggerer	Stages
passed 00:03:04 3 hours ago	Merqe de origen/master #680349845 prod/kantar latest	7ff0cca7	✓✓
passed 00:03:01 3 hours ago	Merqe de origen/master #680349810 prod/servir latest	234718b5	✓✓
passed 00:03:04 3 hours ago	Merqe de origen/master #680349787 prod/sva latest	f17668e4	✓✓
passed 00:02:58 3 hours ago	Merqe de origen/master #680349768 prod/servinova latest	4c861028	✓✓
passed 00:02:57 3 hours ago	Merqe de origen/master #680349724 prod/aitalegal latest	b614fad8	✓✓

### 7.1.1. Métricas de GitLab Analytics

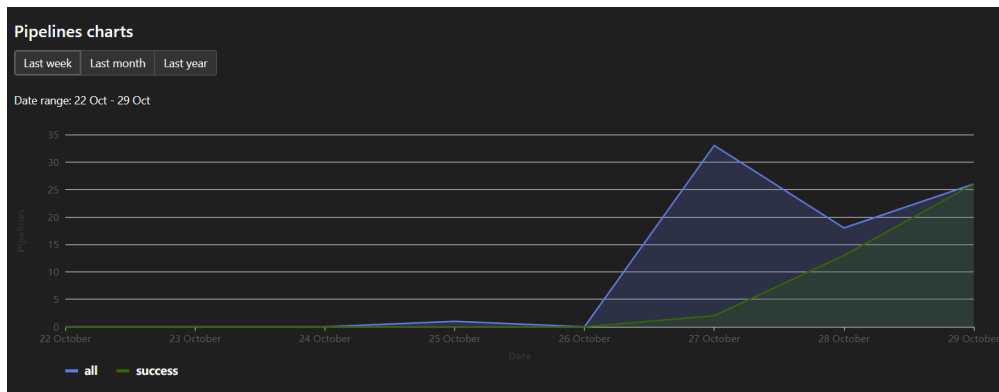
La herramienta de GitLab proporciona a sus usuarios algunas métricas sobre las pipelines ejecutadas del proyecto, la cantidad de pipelines, pipelines fallidas, pipelines finalizadas exitosamente, tiempo de duración de pipeline entre otras.

Figura 20: GitLab CI/CD Analytics - pipelines ejecutadas, fallidas y finalizadas correctamente.



Asimismo es posible visualizar un gráfico de las pipelines ejecutadas ya sea en la última semana, mes o año.

Figura 21: GitLab CI/CD Analytics - gráfico de pipelines ejecutadas en la última semana.



## 7.2. Mediciones de tiempo utilizando GitLab CI/CD y Deployer

Para comprobar el impacto que tiene el proyecto en la automatización de la creación y actualización de instancias de la plataforma web se realizaron nuevas mediciones del tiempo que le toma a una persona poder ejecutar este trabajo utilizando el proyecto como herramienta. A continuación se detallan los resultados obtenidos:

Cuadro 5: Tiempos creación de instancias de instalaciones. Registros de tiempo utilizando GitLab CI/CD y Deployer para crear nuevas instalaciones.

Instalación	Tiempo (horas)
Prueba 1	0.13
Prueba 2	0.10
Prueba 3	0.10
Prueba 4	0.10
Prueba 5	0.10

Cuadro 6: Tiempos actualización de instalaciones. Registros de tiempo utilizando GitLab CI/CD y Deployer para actualizar múltiples instalaciones.

Instalación	Tiempo (horas)
General	0.69
General	1.05
General	0.45
General	0.51
General	0.39

Figura 22: Gráfica de área de mediciones de tiempo para la creación de nuevas instancias de la plataforma

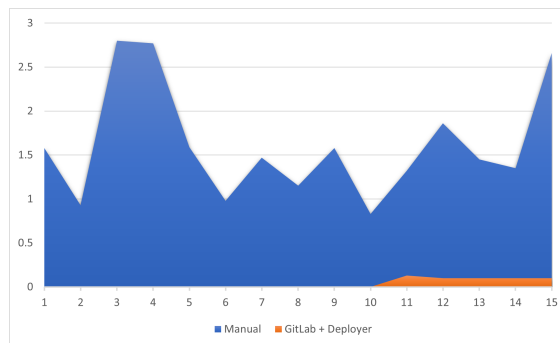


Figura 23: Gráfica de caja y bigotes para mediciones de tiempo para la creación de nuevas instancias de la plataforma

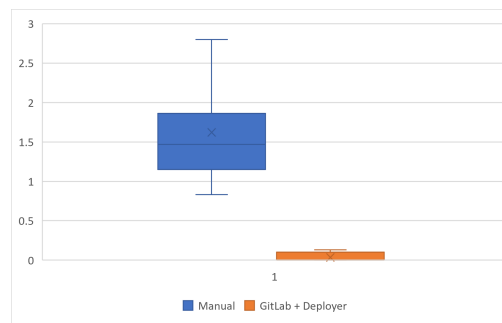


Figura 24: Gráfica de área de mediciones de tiempo para la actualización de un conjunto de instancias de la plataforma

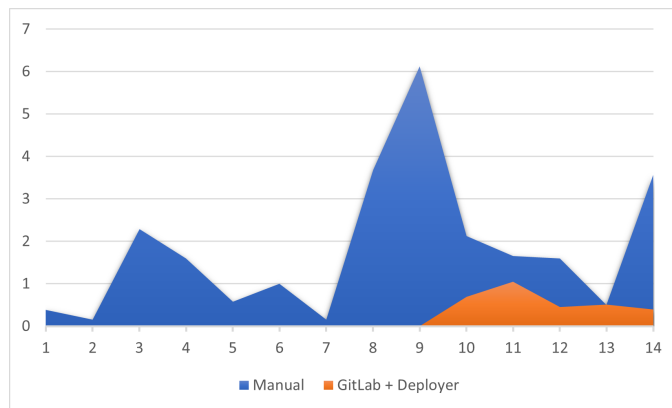
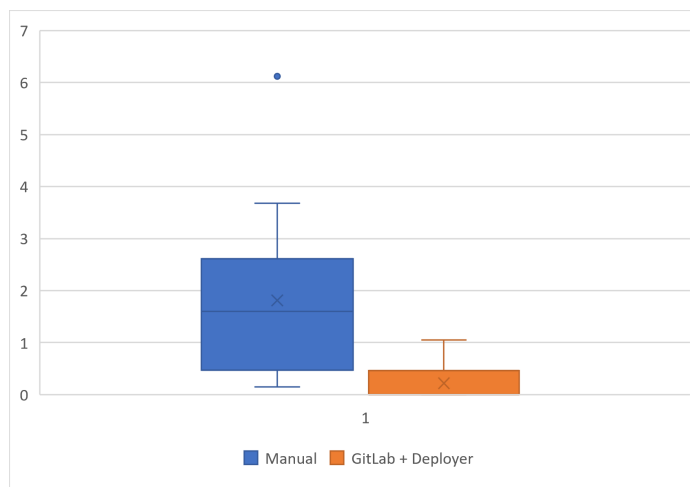


Figura 25: Gráfica de caja y bigotes para mediciones de tiempo para la actualización de un conjunto de instancias de la plataforma



A partir de los gráficos, se puede visualizar que los tiempos utilizando GitLab CI/CD con Deployer tienen una mediana más baja que realizarlo de forma manual. Los datos de los tiempos manuales tienen un sesgo superior. En el caso de los tiempos de actualización de forma manual presentan un dato atípico.

Los tiempos promedios resultantes de las mediciones de tiempo son los siguientes:

Cuadro 7: Tiempo promedio en horas de creación y actualización de la plataforma web de forma manual y automatizada.

Forma	Crear instalación	Actualizar instalaciones
Manual	1.6213 horas	1.8136 horas
Automatizada (GitLab CI/CD y Deployer)	0.1060 horas	0.6180 horas

---

### Análisis de resultados

---

La migración a git permite la utilización de mejores herramientas disponibles en el mercado (como GitLab). Asimismo, la introducción del concepto de docker al flujo de trabajo de los equipos de desarrollo abre nuevas posibilidades de pruebas, tales como probar nuevas versiones del lenguaje de programación o del motor de bases de datos.

Las actividades de entrega de software tienden a incrementarse en la medida que se crea y evoluciona el software dentro de la empresa, por lo cual su automatización es necesaria. Este proyecto fue una prueba de concepto para crear ese proceso rápido, repetible y confiable de entrega de software. Este ciclo también impacta de manera positiva la productividad y eficiencia de los equipos, permitiéndoles enfocarse en temas aún más relevantes para el contexto del negocio.

El despliegue en producción es una operación de cuidado dado que una falla puede ocasionar interrupciones en el servicio a los clientes, por tanto, un ciclo de entrega de software automatizado no solo tiene un efecto positivo en la velocidad sino en la confiabilidad y estandarización del proceso, lo que permite soportar más instalaciones con eficacia.

No se midió la cantidad de bugs existentes, pero se introdujo el desarrollo de pruebas automatizadas, lo que esperamos mejore la detección temprana de errores, aumente la confiabilidad y permita a la empresa introducir mejores prácticas de ingeniería de software.



---

### Conclusiones

---

- El tiempo promedio de creación de nuevas instancias y actualización de la plataforma web sin utilizar procesos automatizados es de 1.6213 y 1.8136 horas respectivamente.
- La herramienta seleccionada para utilizar en el proyecto fue GitLab CI/CD en conjunto con Deployer.
- El desarrollo de pruebas automatizadas se realizó utilizando la herramienta de PHPUnit.
- La implementación del proceso de CI y CD implicó la migración de Subversion a GitLab, la creación de una imagen de Docker, creación de recetas en Deployer y la configuración de GitLab CI/CD.
- El pipeline desarrollada está conformada por dos etapas: *test* y *deploy*.
- La utilización de GitLab CI/CD y las recetas de Deployer abarcan los procesos de creación de instancias y actualización de la plataforma web.
- El tiempo promedio de creación de nuevas instancias y actualización de la plataforma web utilizando GitLab CI/CD y Deployer es de 0.1060 y 0.6180 horas respectivamente.
- La fase de creación de nuevas instancias es en promedio de aproximadamente 15 veces más ágil y el proceso de actualización es de casi 3 veces más rápido de realizar.





## CAPÍTULO 10

---

### Recomendaciones

---

- Este proyecto se desarrolló específicamente para entornos Linux, por lo que se recomienda implementarlo en distintos ambientes para corroborar su funcionalidad.
- Implementar la pipeline automatizada para distintos frameworks y lenguajes de programación para verificar que los resultados son consistentes para distintos ambientes de trabajo.
- Evaluar la utilización de herramientas que contemplen Ansible para la automatización completa del manejo de infraestructura de los sistemas de software y procedimientos de TI de una empresa.
- Implementar al proyecto una forma de integración para detección y notificación de errores durante el proceso.
- Agregar soporte para ejecución de otros tipos de pruebas automatizadas al pipeline (pruebas de integración, pruebas de extremo a extremo, etc.)



---

## Referencias de la red o de internet

---

- Atlassian. (2022). ¿Qué es el control de versiones? [Recuperado 28 de octubre de 2022]. <https://www.atlassian.com/es/git/tutorials/what-is-version-control>
- CDNNetworks. (2022). 5 Statistics that Prove DevOps Practices Improve Web Performance [Recuperado 22 de marzo de 2022]. <https://www.cdnetworks.com/web-performance-blog/5-statistics-that-prove-devops-practices-improve-web-performance/>
- Dee, K. (2021). Report: CI/CD still in early phase for much of software development community [Recuperado 23 de marzo de 2022]. <https://sdtimes.com/softwaredev/report-ci-cd-still-in-early-phase-for-much-of-software-development-community/>
- Encora. (2020). Integración continua e implementación continua en GitLab. [Recuperado 21 de septiembre de 2022]. <https://www.encora.com/es/blog/implementing-continuous-integration-and-continuous-deployment-on-gitlab>
- G2. (2022). GitLab [Recuperado 24 de octubre de 2022]. <https://www.g2.com/products/gitlab/reviews>
- GitLab. (2022). Slack notifications integration [Recuperado 24 de octubre de 2022]. <https://docs.gitlab.com/ee/user/project/integrations/slack.html>
- Grup, I. (2018). Cinco formas de crear una cultura DevOps [Recuperado 3 de enero de 2023]. <https://idgrup.com/cinco-formas-de-crear-una-cultura-devops-de-alto-rendimiento/>
- Hristov, A. (2021). Cómo las pruebas automatizadas hacen posible DevOps [Recuperado 28 de octubre de 2022]. <https://www.atlassian.com/es/devops/devops-tools/test-automation>
- IONOS. (2020). Git vs. SVN: ¿cuál es el mejor sistema de control de versiones? [Recuperado 28 de octubre de 2022]. <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/git-vs-svn-una-comparativa-del-control-de-versiones/>
- Jain, R. (2020). Jenkins vs GitLab CI: Battle of CI/CD Tools [Recuperado 24 de octubre de 2022]. <https://www.lambdatest.com/blog/jenkins-vs-gitlab-ci-battle-of-ci-cd-tools/>
- Khatri, V. (2022). Las 19 mejores herramientas de prueba de software [Recuperado 3 de enero de 2023]. <https://geekflare.com/es/software-testing-tools/>
- Medvedev, A. (2013). Deployer [Recuperado 1 de agosto de 2022]. <https://deployer.org/>
- Medvedev, A. (2022). Deployer Packagist [Recuperado 23 de octubre de 2022]. <https://packagist.org/packages/deployer/deployer>

- NetApp. (2019). ¿Qué es DevOps? [Recuperado 3 de septiembre de 2022]. <https://www.netapp.com/es/devops-solutions/what-is-devops/>
- Oracle. (2021). ¿Qué es Docker? [Recuperado 23 de octubre de 2022]. <https://www.oracle.com/mx/cloud/cloud-native/container-registry/what-is-docker/>
- PHPUnit. (2021). PHPUnit [Recuperado 23 de octubre de 2022]. <https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/273>
- Santiago Montoya, M. A. O. (2018). What is CI/CD? [Recuperado 3 de mayo de 2022]. <https://www.redhat.com/en/topics/devops/what-is-ci-cd>
- Santiago Montoya, M. A. O. (2020). Conoce 5 herramientas para integración y entrega continua con DevOps [Recuperado 10 de mayo de 2022]. <https://www.pragma.com.co/blog/conoce-5-herramientas-para-integracion-y-entrega-continua-con-devops>
- Sentry. (2022). Gitlab + Sentry Integration [Recuperado 24 de octubre de 2022]. <https://sentry.io/integrations/gitlab/>
- Taylor, D. (2022). What is DevOps? Full Form, Meaning, Principles & Examples [Recuperado 3 de septiembre de 2022]. <https://www.guru99.com/what-is-devops.html>
- Wesley Chai, K. C. (2021). Software as a Service (SaaS) [Recuperado 23 de mayo de 2022]. <https://www.techtarget.com/searchcloudcomputing/definition/Software-as-a-Service>

### 12.1. Creación de una cultura de DevOps

DevOps representa un cambio cultural. No se trata simplemente de adoptar una metodología ágil de planificación, pruebas automatizadas o entrega continua, aunque esas prácticas son, sin ninguna duda, importantes. La cultura de DevOps implica que los desarrolladores y los profesionales de operaciones compartan una misma visión y la responsabilidad por el software que crean. Implica aumentar la transparencia, la comunicación y la colaboración en los equipos de desarrollo, operaciones/TI y empresariales (Grup, 2018).

- **Cultura que adopta una mentalidad de pensamiento sistémico:** una organización de DevOps debe considerar el sistema como el negocio mismo. El establecimiento de prácticas de DevOps no debe estar aislado y centrado en disciplinas, equipos o departamentos específicos. El pensamiento sistémico significa que cada equipo debe conocer las acciones de cada equipo en el ciclo de vida de la aplicación y los resultados de cada acción interna y externamente. El sistema trabaja hacia objetivos comunes y se mide de manera integral. Una cartera de entrega continua reúne muchas prácticas, procesos y procedimientos diferentes con un objetivo común: entregar software de alta calidad con riesgo reducido y mayor velocidad (Grup, 2018).
- **Cultura que cree que la calidad es una responsabilidad compartida:** la calidad que es propiedad exclusiva de un equipo de control de calidad es un claro anti-patrón de DevOps. En los paradigmas modernos de la entrega de aplicaciones, la calidad debe convertirse en un trabajo de flujo de entrega continuo y, por lo tanto, una responsabilidad compartida en los equipos de desarrollo, prueba, lanzamiento y operaciones. El simple hecho de centrarse en la aceleración de la entrega de la aplicación a menudo pasa por alto la marca. Debe llegar al punto de calidad de ingeniería en su software a través de pruebas continuas. El objetivo de las pruebas continuas es instrumentar la calidad en su canal de entregas desplazándose a la izquierda para solucionar problemas tan pronto como se introducen y eliminando los

problemas antes de que se conviertan en una emergencia de primer nivel en la producción. Esto ayuda a que la tubería se mueva más eficientemente y mantiene la armonía (Grup, 2018).

- **Cultura que fomenta la experimentación:** la capacidad de experimentar, aprender, fracasar, repetir, es crucial para implementar una metodología DevOps exitosa. La experimentación comienza con la confianza, que se puede establecer haciendo coincidir a todos en la misma página con las prácticas y criterios comunes para mover una versión candidata a través de la canalización. Los equipos deben reconocer, compartir y celebrar sus aprendizajes a prueba de fallas de manera similar a como celebramos los éxitos. Es igualmente importante compartir su conocimiento con otros equipos dentro de la organización para que puedan evolucionar sus procesos a lo largo del camino (Grup, 2018).
- **Cultura que continuamente toma decisiones basadas en datos:** es esencial medir y monitorear el progreso en cada etapa del ciclo de vida de la aplicación, tal como lo haría con cualquier otra iniciativa comercial. Con demasiada frecuencia no hay datos, o los datos son demasiado inconsistentes entre los equipos para poder entender lo que funciona o lo que necesita funcionar. Todos los equipos deben aprovechar las mismas fuentes de datos y creer en los datos. Los análisis de DevOps permiten que los equipos demuestren evidencia tangible de progreso, se centren en áreas que necesitan mejoras y tomen los pasos en la dirección correcta juntas (Grup, 2018).

## 12.2. Herramientas de pruebas automatizadas

- **Apache JMeter:** una de las herramientas para pruebas de bases de datos que funcionen con páginas web, escrito completamente en lenguaje Java y orientado directamente a los programadores. Sirve probar la carga masiva de datos y realizar pruebas de esfuerzo a los servidores, debemos utilizar JMeter con un software de monitorización como Pandora FMS. Apache JMeter no es un navegador web (no ejecuta el JavaScript del código HTML, por ejemplo) sino que actúa a nivel de protocolo HTTP, HTTPS (Java, NodeJS, PHP, ASP.NET, entre otros), FTP, SMTP(S), POP3(S) e IMAP(S) (Khatri, 2022).
- **Selenium:** Es una herramienta portátil de código abierto y proporciona pruebas funcionales de varios módulos de aplicaciones web en plataformas y navegadores. Admite la ejecución de texto en paralelo, lo que ahorra mucho tiempo de prueba. Corre en Microsoft Windows, GNU/Linux y OS X; soporta Java, Groovy, Python, C#, PHP, Ruby, y Perl; es una de las herramientas para pruebas de aplicaciones web que se integra a Chrome, Firefox, Microsoft IE e incluso navegadores sin interfaz gráfica. Los componentes de Selenium son: IDE, Client API, Remote Control, WebDriver y Grid. Admite integración con Jenkins y Cruise Control (Khatri, 2022).
- **Cucumber:** es una herramienta de prueba de código abierto que está escrita en el lenguaje Ruby. Esta herramienta funciona mejor para el desarrollo impulsado por el comportamiento. Pero también se puede usar para probar otros lenguajes como Java, C# y Python (Khatri, 2022).
- **Gatling:** esta herramienta de prueba de código abierto está especialmente diseñada para CI/CD y DevOps para pruebas de carga. Con Gatling, puede evitar bloqueos y tiempos de respuesta lentos. Detecta rápidamente los errores y problemas en el rendimiento de su aplicación

durante el desarrollo. Como resultado, puede obtener una imagen precisa de la experiencia del usuario (Khatri, 2022).

- **Testim:** es una herramienta de prueba de software automatizada inteligente que utiliza el aprendizaje automático para acelerar el diseño, la ejecución y el mantenimiento de casos de prueba automatizados. Los casos de prueba se pueden ejecutar en múltiples plataformas, incluidos dispositivos móviles. Testim usa anotaciones para encontrar inconsistencias y errores en el sistema. Los errores que se registran se pueden reproducir automáticamente simplemente haciendo clic en la prueba nuevamente. El rastreador de errores Testim se usa para compartir capturas de pantalla anotadas y ver detalles de errores (Khatri, 2022).
- **Xray:** es una de las herramientas de gestión de pruebas preferidas para pruebas manuales y automatizadas. Proporciona la estructura adecuada para organizar y clasificar conjuntos de pruebas y proporciona resultados de prueba eficientes en menos tiempo. Se integra perfectamente con marcos de prueba como Jira, Selenium, Junit, etc. Puede establecer condiciones previas personalizadas que se pueden reutilizar y asociar con diferentes pruebas. Xray facilita las integraciones de CI con Bamboo, Jenkins y proporciona una trazabilidad perfecta y una correspondencia entre requisitos, pruebas, errores y ejecución. También tiene disposiciones para establecer entornos de prueba, planes de prueba y la generación de informes integrados (Khatri, 2022).

