



Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

**INDUSTRIALIZACIÓN DE UNA APLICACIÓN WEB EN UN AMBIENTE CLOUD
IMPLEMENTANDO DEVOPS CI/CD Y DELIVERY HACIA EL CLIENTE**

Luis Fernando Pérez Morán

Asesorado por el Ing. José Roberto Cámara Bran

Guatemala, septiembre de 2022

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA



FACULTAD DE INGENIERÍA

**INDUSTRIALIZACIÓN DE UNA APLICACIÓN WEB EN UN AMBIENTE CLOUD
IMPLEMENTANDO DEVOPS CI/CD Y DELIVERY HACIA EL CLIENTE**

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

LUIS FERNANDO PÉREZ MORÁN

ASESORADO POR EL ING. JOSÉ ROBERTO CÁMARA BRAN

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, SEPTIEMBRE DE 2022

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA



NÓMINA DE JUNTA DIRECTIVA

DECANA	Inga. Aurelia Anabela Cordova Estrada
VOCAL I	Ing. José Francisco Gómez Rivera
VOCAL II	Ing. Mario Renato Escobedo Martínez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Br. Kevin Vladimir Armando Cruz Lorente
VOCAL V	Br. Fernando José Paz Gonzáles
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANA	Inga. Aurelia Anabela Cordova Estrada
EXAMINADOR	Ing. César Augusto Fernández Cáceres
EXAMINADOR	Ing. César Rolando Batz Saquimux
EXAMINADOR	Ing. Álvaro Giovanni Longo Morales
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

INDUSTRIALIZACIÓN DE UNA APLICACIÓN WEB EN UN AMBIENTE CLOUD IMPLEMENTANDO DEVOPS CI/CD Y DELIVERY HACIA EL CLIENTE

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha septiembre 2022.

Luis Fernando Pérez Morán

Guatemala, 08 de junio de 2022

Ingeniero
Carlos Alfredo Azurdia
Coordinador de Privados y Trabajos de Tesis
Escuela de Ingeniería en Ciencias y Sistemas
Facultad de Ingeniería - USAC

Respetable Ingeniero Azurdia:

Por este medio hago de su conocimiento que en mi rol de asesor del trabajo de investigación realizado por el estudiante **LUIS FERNANDO PÉREZ MORÁN** con carné 201212822 y CUI 2221 89916 0101 titulado "INDUSTRIALIZACION DE UNA APLICACIÓN WEB EN UN AMBIENTE CLOUD IMPLEMENTANDO DEVOPS CI/CD Y DELIVERY HACIA EL CLIENTE", luego de corroborar que el mismo se encuentra finalizado, lo he revisado y doy fe de que el mismo cumple con los objetivos propuestos en el respectivo protocolo, por consiguiente, procedo a la aprobación correspondiente.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,



Ing. José Roberto Cámara Bran
Colegiado No. 15054

José Roberto Cámara Bran
INGENIERO EN CIENCIAS Y SISTEMAS
Colegiado No. 15,054



Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala 6 de julio de 2022

Ingeniero
Carlos Gustavo Alonzo
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Alonzo:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **LUIS FERNANDO PÉREZ MORÁN** con carné **201212822** y CUI **2221 89916 0101** titulado **"INDUSTRIALIZACIÓN DE UNA APLICACIÓN WEB EN UN AMBIENTE CLOUD IMPLEMENTANDO DEVOPS CI/CD Y DELIVERY HACIA EL CLIENTE"** y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo aprobado.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación



UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA



FACULTAD DE INGENIERÍA

LNG.DIRECTOR.164.EICCSS.2022

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del Asesor, el visto bueno del Coordinador de área y la aprobación del área de lingüística del trabajo de graduación titulado: **INDUSTRIALIZACIÓN DE UNA APLICACIÓN WEB EN UN AMBIENTE CLOUD IMPLEMENTANDO DEVOPS CI/CD Y DELIVERY HACIA EL CLIENTE**, presentado por: **Luis Fernando Pérez Morán**, procedo con el Aval del mismo, ya que cumple con los requisitos normados por la Facultad de Ingeniería.

"ID Y ENSEÑAD A TODOS"



Msc. Ing. Carlos Gustavo Alonzo
Director
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, septiembre de 2022





Decanato
Facultad de Ingeniería
24189101- 24189102
secretariadecanato@ingenieria.usac.edu.gt

LNG.DECANATO.OI.583.2022

La Decana de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al Trabajo de Graduación titulado: **INDUSTRIALIZACIÓN DE UNA APLICACIÓN WEB EN UN AMBIENTE CLOUD IMPLEMENTANDO DEVOPS CI/CD Y DELIVERY HACIA EL CLIENTE**, presentado por: **Luis Fernando Pérez Morán**, después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:


Inga. Aurelia Anabela Cordova Estrada

Decana

Guatemala, septiembre de 2022

ACTO QUE DEDICO A:

Dios	Por ser el dador del conocimiento y quien me ayudo a sobrellevar cada momento en mi carrera.
Mis padres	Luis Pérez y Aleyda Morán por siempre confiar en mí y permitirme seguir mis sueños.
Mi esposa	Claudia Chávez por estar en cada desvelo a mi lado y darme alientos cuando veía muy lejano la meta.
Mi hija	Mia Lucia Pérez por ser el ángel que me inspiro.
Mis hijos	Geovany y Alexander Villatoro por su ayuda y apoyo en todo momento
Mis amigos	Por ser parte de esta ardua carrera.
Ingeniero	José Cámara por ser fuente de conocimiento en el desarrollo de mi tesis.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala Por ser esa *alma máter* que me formo.

Mis amigos Por el apoyo en las largas noches de desvelos

Ingenieros Que durante de la carrera fueron fuente de conocimiento y aprendizaje

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES.....	V
GLOSARIO	IX
RESUMEN.....	XI
OBJETIVOS.....	XIII
INTRODUCCIÓN	XV
1. MARCO TEORICO.....	1
1.1. ¿Qué es <i>DevOps</i> ?	1
1.2. <i>Continuous Integration Delivery (CI/CD)</i>	2
1.2.1. <i>Continuous Integration (CI)</i>	2
1.2.2. <i>Continuous Deployment</i>	3
1.2.3. <i>Continuous Delivery (CD)</i>	3
1.3. Fases del proceso <i>DevOps</i>	4
1.3.1. Planeación y diseño.....	5
1.3.2. Creación	6
1.3.3. Construcción.....	8
1.3.4. Calidad	8
1.3.5. Despliegue.....	9
1.3.6. Verificación y Liberación.....	10
1.3.7. Monitoreo.....	11
1.4. Metodologías ágiles.....	12
1.4.1. <i>Scrum</i>	13
1.5. Herramientas para la gestión de proyectos	14
1.5.1. <i>Jira</i>	15
1.5.2. <i>Confluence</i>	16

1.6.	Herramientas para la gestión de código.....	17
1.6.1.	<i>GitLab</i>	18
1.7.	Servicios en la nube	19
1.7.1.	Proveedores de servicios en la nube	20
1.7.2.	<i>Software</i> Como Servicio (SaaS).....	22
1.7.3.	Infraestructura como Servicio (IaaS)	23
1.7.4.	Plataforma Como Servicio (PaaS).....	23
1.7.5.	Servicios en la nube para <i>DevOps</i>	24
1.8.	<i>React</i>	25
1.8.1.	Origen de <i>React</i>	26
1.8.2.	Desarrollo de aplicaciones <i>web</i> basadas en componentes.....	27
2.	COMPARATIVA DE HERRAMIENTAS EN LA IMPLEMENTACIÓN DE <i>DEVOPS</i>	29
2.1.	Herramientas de planeación y diseño	29
2.1.1.	Comparativa de <i>Jira</i> vs <i>Trello</i>	29
2.1.2.	Comparativa <i>Xray</i> vs <i>Kualitee</i>	32
2.1.3.	Comparativa <i>Portfolio Jira</i> vs <i>Microsoft Project</i>	35
2.1.4.	Comparativa <i>Confluence</i> vs <i>AnswerHub</i>	38
2.1.5.	Conclusión de herramientas de planeación y diseño.....	40
2.2.	Herramientas de creación	41
2.2.1.	Comparativa <i>GitLab CI</i> vs <i>Jenkins</i>	41
2.3.	Herramientas de calidad	45
2.4.	Herramientas de despliegue	47
2.4.1.	Comparativa <i>Ansible</i> vs <i>Fabric</i>	48
2.5.	Herramientas de verificación y liberación.....	51
2.5.1.	Comparativa <i>SoapUI</i> vs <i>Postman</i>	52

2.6.	Lenguajes de programación orientados a componentes	55
2.6.1.	Comparativa <i>React js</i> vs <i>Angular</i>	55
2.6.1.1.	Desarrollo para móviles	56
2.6.1.2.	Curva de aprendizaje.....	56
2.6.1.3.	Desempeño	57
2.6.1.4.	Comentario	57
3.	ESTÁNDARES DE PROCESOS	59
3.1.	Manejo de documentación en <i>Confluence</i>	59
3.2.	Creación de Épicas, historias de usuarios y tareas en <i>Jira</i>	62
3.3.	Manejo correcto de ramas	65
3.3.1.	Nomenclatura correcta para ramas	66
3.3.2.	Nomenclatura correcta para <i>commit</i> en el proyecto.....	69
3.3.3.	Utilización de la función <i>rebase</i> en las ramas periódicamente	71
3.4.	Definición de parámetros para aceptación de la calidad de código en <i>SonarQube</i>	73
3.4.1.	<i>Quality Profile</i>	73
3.4.2.	<i>Quality Gate</i>	74
4.	IMPLEMENTACION DE PROCESO <i>DEVOPS</i>	77
4.1.	Planeación y diseño	77
4.1.1.	Creación de espacio en <i>Jira</i>	77
4.1.2.	Creación de <i>confluence</i>	81
4.1.3.	Creación de repositorio en <i>GitLab</i>	84
4.1.4.	Configuración de <i>runner</i> en <i>GitLab</i>	87
4.1.5.	Creación de tareas de desarrollo y <i>planning</i>	90
4.1.5.1.	Creación de Épica	90

	4.1.5.2.	Creación de Historia de usuario	91
	4.1.5.3.	Utilización de <i>planning</i> de <i>Jira</i>	93
	4.1.6.	Creación de reportes para control de avances.....	94
	4.1.7.	Creación de <i>Kanban</i> de tareas.....	96
4.2.		Creación.....	98
	4.2.1.	Configuración de <i>IDE</i> de desarrollo.....	99
	4.2.2.	Creación de fuentes iniciales.....	101
	4.2.3.	Creación de rama para desarrollo	103
4.3.		Construcción	105
	4.3.1.	Creación de <i>stage</i> de <i>build</i> en <i>pipeline</i>	106
4.4.		Calidad.....	108
	4.4.1.	Creación de <i>stage</i> de prueba en <i>pipeline</i>	108
	4.4.2.	Creación de <i>stage</i> de <i>scan code</i> en <i>pipeline</i>	110
4.5.		Despliegue	111
	4.5.1.	Creación de <i>stage</i> de <i>deploy</i>	112
4.6.		Verificación.....	115
	4.6.1.	Creación de <i>stage</i> de <i>automatic test</i>	115
	4.6.2.	Creación de <i>stage performance</i>	117
5.		PLANTILLAS.....	119
	5.1.	Plantilla de estructura de nuevo proyecto en <i>jira</i>	119
	5.2.	Plantilla de <i>gitlab CI</i>	121
	5.3.	Plantilla de <i>ansible</i>	124
		CONCLUSIONES.....	127
		RECOMENDACIONES	129
		REFERENCIAS	131
		APÉNDICES.....	135

ÍNDICE DE ILUSTRACIONES

FIGURAS

1.	<i>Continuous Delivery</i>	4
2.	Fases del proceso <i>DevOps</i>	5
3.	Cuadrante Mágico de <i>Gartner 2021</i>	20
4.	<i>GitFlow</i>	66
5.	Imagen de épica.....	68
6.	Creación de rama.....	68
7.	<i>Commit</i> enlazado con <i>Jira</i>	70
8.	<i>Rebase</i> en <i>git</i>	72
9.	Tablero de <i>SonarQube</i>	75
10.	<i>Workflow SonaQube</i>	76
11.	Creación de sitio <i>Jira</i>	78
12.	Ayudante de configuración.....	79
13.	Plantilla inicial.....	79
14.	Proyecto en <i>Jira</i>	80
15.	<i>Confluence</i>	82
16.	Página inicial <i>Confluence</i>	83
17.	Plantilla de manejo de documentación.....	84
18.	Pantalla de inicio de sesión y creación de cuenta.....	85
19.	Pantalla de creación de proyecto	86
20.	Pantalla de proyecto en <i>GitLab</i>	87
21.	<i>Runners</i> de <i>gitlab</i>	89
22.	Creación de Épica.....	91
23.	Creación de Historia de usuario	92

24.	<i>Planning de jira</i>	93
25.	Creación de panel para reportes.....	94
26.	Panel de reportería	95
27.	<i>Backlog jira</i>	97
28.	<i>Kanban sprint 1</i>	98
29.	<i>Visual Code</i>	100
30.	<i>Templates de PrimeReact</i>	102
31.	Clonado de repositorio.....	104
32.	Creación de <i>stage de build</i>	107
33.	Creación de <i>stage de prueba</i>	109
34.	Archivo ejecución <i>sonar</i>	110
35.	Configuración escaneo de código <i>pipeline</i>	111
36.	Configuración de <i>Google Cloud run</i>	113
37.	Configuración de <i>Google Cloud run</i>	114
38.	Creación de <i>job de test automation</i>	116
39.	Creación de <i>job de performance</i>	118

TABLAS

I.	Comparativa <i>Jira vs Trello</i>	30
II.	Comparativa <i>Xray vs Kualitee</i>	32
III.	Comparativa <i>Porfolio Jira vs Microsoft Project</i>	35
IV.	Comparativa <i>Confluence vs AnswerHub</i>	38
V.	Comparativa <i>GitLab CI vs Jenkins</i>	42
VI.	Comparativa <i>SonarQube vs Kiuwan</i>	45
VII.	Comparativa <i>Ansible vs Fabric</i>	48
VIII.	Comparativa <i>SoapUI vs Postman</i>	52
IX.	Plantilla de propiedades <i>Jira</i>	119
X.	Datos necesarios para plantilla.....	121

XI. Datos plantilla..... 123
XII. Datos de plantilla *ansible* 124

GLOSARIO

<i>Continuous Delivery</i>	Enfoque de la ingeniería del <i>software</i> en que los equipos de desarrollo producen <i>software</i> en ciclos cortos, garantizando que la entrega de los artefactos se realice en cualquier momento.
<i>Continuous Deployment</i>	Parte del proceso de <i>DevOps</i> en el cual los equipos de desarrollo realizan entregas continuas de nuevas funcionalidades de manera periódica.
<i>Continuous Integration</i>	Práctica de la ingeniería de <i>software</i> que consiste en hacer integraciones automáticas de un proyecto lo más frecuentemente posible para así poder detectar fallos cuanto antes.
<i>DevOps</i>	Es un conjunto de prácticas que agrupan el desarrollo de <i>software</i> (<i>Dev</i>) y las operaciones de <i>TI</i> (<i>Ops</i>).
<i>GitLab</i>	Servicio <i>web</i> de control de versiones y desarrollo de <i>software</i> colaborativo basado en git el cual nos brinda una gama de herramienta con las cuales podemos hacer una implementación de <i>CI/CD</i> dentro de ella.
Metodología	Conjunto de procedimientos relacionales utilizados para alcanzar un objetivo.

Scrum

Es un marco de trabajo para el desarrollo ágil de *software*, en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo y obtener el mejor resultado posible.

RESUMEN

El presente trabajo de tesis pretende el exponer como utilizar estándares de la industria de la ingeniería de *software* y crear una estructura inicial para poder implementar un *pipeline* en la parte de integración continua y despliegue continuo conocido como *CI/CD* en un ambiente *Cloud* de manera estándar y genérica así como ir adoptando una cultura de *DevOps* en la cual varios roles interactúan y es importante el saber cómo cada uno de estos deben interactuar y cuales son la tareas que cada uno debe realizar en función de poder tener una línea de ensamblaje de *software* con la cual se pueda agilizar la liberación de futuras versiones de *software*.

Se desarrolló también una serie de plantillas que pueden utilizarse para poder realizar la implementación del proceso de *DevOps*. Además, se establecieron estándares y buenas prácticas que se deben ir implementando en cada una de las etapas del proceso.

Además, de realizar una comparativa entre las herramientas existentes para el proceso *DevOps* e ir exponiendo las razones por las cuales se prefiere la utilización de una herramienta en particular sobre otra.

OBJETIVOS

General

Establecer cuáles son los mejores estándares para implementar en un proceso de *DevOps*, mediante la documentación del proceso y a su vez la creación de plantillas, para facilitar y estandarizar la implementación en cualquier ambiente *Cloud*.

Específicos

1. Definir estándares de cómo implementar el proceso de *DevOps* y como irlo documentando en cada una de sus fases.
2. Establecer plantillas que puedan ser reutilizables en las herramientas con el fin de agilizar la implementación en un ambiente *Cloud*.
3. Comparar las diferentes herramientas utilizadas en las diversas fases del proceso *DevOps* para identificar bajo que circunstancia es conveniente utilizar cada una.

INTRODUCCIÓN

A lo largo del tiempo, la tecnología ha ido evolucionando y con ello los procesos que se utilizan para desarrollar dichas tecnologías, es por ello por lo que el desarrollo de las aplicaciones y la manera de crear *software* también ha ido evolucionando a tal punto que con el pasar del tiempo fueron surgiendo diferentes metodologías como cascada, prototipado, espiral, ágiles, entre otras. En donde se habla de las metodologías ágiles, estas han tenido un auge en el desarrollo de *software* en los últimos años.

DevOps es una forma de hacer *software* de manera rápida, ágil y con la menor cantidad de errores posibles. Y es por ello por lo que se desarrolló una forma estándar de ir implementando el proceso de *DevOps* en cada una de sus fases y tener las diversas ventajas que brinda el desarrollar *software* utilizando esta metodología.

En este documento de tesis se encontrarán diversas plantillas que podrán ayudar a implementar de una mejor manera el proceso de *DevOps* junto a una serie de estándares de cómo ir manejando herramientas, *software* y documentación. Por último, también se tendrá una serie de comparativas de las diversas herramientas que se pueden utilizar en cada una de las fases del proceso para conocer fortalezas, debilidades o incluso bajo que circunstancia escoger una sobre las otras.

1. MARCO TEORICO

1.1. ¿Qué es *DevOps*?

DevOps es un modo de abordar la cultura de la automatización, es la unión de personas, procesos y tecnologías para ofrecer valor a los clientes de forma constante y con mayor capacidad de respuesta. El término “*DevOps*” es una combinación de las palabras *development* (desarrollo) y *operations* (operaciones), pero no solo abarca estos dos conceptos sino en ello encontramos un conjunto de ideas y prácticas que han hecho de esta metodología lo que es hoy (Azure, 2021).

Esta metodología ayuda a unificar los roles que se encontraban separados en un proceso de desarrollo de *software* tradicional, ahora roles como desarrollador, control de calidad, seguridad, operaciones se coordinen y trabajen en conjunto para poder tener un producto final de una mejor calidad en un tiempo más corto con la menor cantidad de errores posibles.

Al implementar este proceso desde la fases de análisis y diseño de una aplicación brinda muchas ventajas que se verán reflejadas en el producto final, siendo de las principales características que brinda el fomentar el trabajo en equipo en cada una de las fases, además que al ser también una cultura que van adoptando se inicia a estandarizar el procesos, esto debido a la utilización herramientas propias para la implementación de *DevOps* y que con ellas también se van aplicando una serie de estándares y buenas prácticas (Red Hat, 2018).

1.2. *Continuos Integration Delivery (CI/CD)*

El término *CI/CD* en el habla inglesa hace referencia a *Continuous Integration Continuous Delivery* que en español se traduce como “Integración Continua Despliegue Continuo”, donde se puede agregar el concepto de *Continuous Delivery* el cual corresponde a la entrega de los artefactos finales en un ambiente del cliente mientras que *Continuous Deployment* es en los ambientes propios del equipo de desarrollo. Este concepto representa el eje central del proceso de *DevOps* y el cual se genera nuevas funcionalidades o corregir errores de una manera más rápida ya que se cuenta con varios procesos automatizados que permiten eliminar el factor error humano y que genera respuestas en tiempos más cortos (Pittet, 2021).

Se sabe también que esta parte del proceso de *DevOps* permite la administración de los recursos del sistema, los servidores, máquinas virtuales, bases de datos, contenedores, entre otros. Además, el uso de herramientas de administración de la configuración permite el poder replicar o desplegar un cambio en diversos ambientes para las pruebas o puesta en producción.

1.2.1. *Continuos Integration (CI)*

La integración continua es la práctica de que permite el poder automatizar la integración del código proveniente de diferentes fuentes en un solo proyecto de *software*. Es bien sabido que los proyecto de *software* de gran envergadura suelen necesitar de varios programadores para desarrollar una serie de funcionalidades, donde cada una de estas nuevas funcionalidades pueden llegar a generar cientos de líneas de código en archivos nuevos o ya existentes y es de ahí donde surge unos de los problemas que son resueltos con esta práctica ya que al implementarse la metodología de *DevOps* se fusionan los cambios

existentes con los nuevos en un repositorio central en el cual luego se puede ejecutar el proceso de compilación y de pruebas de manera automatizada (Rehkopf, 2021).

Se puede observar el punto clave de CI es el uso de un sistema de control de versiones el cual permite tener todo el código del proyecto centralizado y que con ello poder administrar y validar cada vez que se integre nuevo código o una nueva funcionalidad esta esté correcta. Esto basándose en temas como la compilación del código y la ejecución de pruebas automatizadas

1.2.2. *Continuous Deployment*

El despliegue continuo como es conocida esta fase del proceso se encuentra centrada en la entrega periódica de *software* con nuevas funcionalidades o correctivos a los ambientes controlados como lo podrían ser *Develop* para el equipo de desarrollo y *Testing* para los encargados de las pruebas de la aplicación.

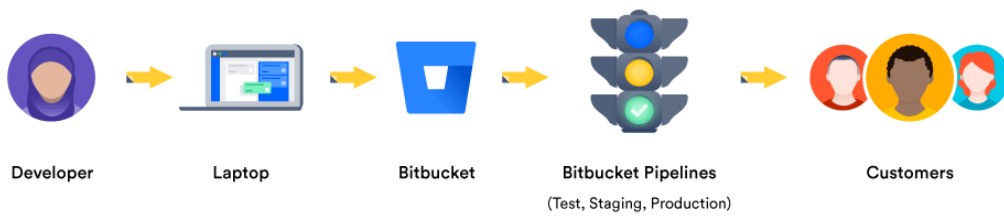
1.2.3. *Continuous Delivery (CD)*

La entrega continua es el paso siguiente al proceso de integración continua y se enfoca en entregar de manera periódica un producto de calidad de manera automática desde el repositorio que centraliza todo nuestro código al ambiente final del cliente.

La entrega continua pasa por una serie de pasos previo a ser puesta en el ambiente destino, en este proceso se observa que se tiene la interacción de un desarrollador que crear el código y es subido al repositorio en la nube, este código

pasa por una serie de *step* como la compilación, la realización de pruebas y posterior la puesta en producción (Atlassian, 2021).

Figura 1. **Continuous Delivery**



Fuente: Atlassian (2021). *¿Qué es la entrega continua?* Consultado el 02 de septiembre de 2021. Recuperado de <https://www.atlassian.com/es/continuous-delivery>.

1.3. Fases del proceso *DevOps*

La metodología *DevOps* se encuentra dividida en diferentes fases que permiten organizar de inicio a fin el proceso de desarrollo y de implementación, por lo que se irán abordando cada una de estas fases en esta sección (Pragma, 2021).

Figura 2. Fases del proceso *DevOps*



Fuente: Academia Pragma (2021). *DevOps: la cultura ágil para la entrega de software*.

Consultado el 02 de septiembre de 2021. Recuperado de <https://www.pragma.com.co/academia/conceptos/devops>.

1.3.1. Planeación y diseño

La primera fase del ciclo de *DevOps* se basa en el desarrollo y creación de lo que serán las futuras funcionalidades por implementar o bien una corrección a realizar sobre un *software* existente. Al ser la etapa que da inicio al ciclo de vida del proceso se toma en cuenta una serie de factores como el involucrar al personal de operaciones y de *TI* en el diseño de los requerimientos, documentar cada uno de los procesos desde un inicio, definir las herramientas a utilizar y no hacerlo sobre la marcha.

Durante esta parte del proceso se pueden apoyar con herramientas para documentar el proceso y definir tiempos procurando que toda esta información

se encuentre centralizada y que sea de fácil acceso para el equipo, pero también con cierto nivel de restricción ya que no es conveniente que todos puedan acceder a toda la información ya que dentro del proceso de diseño puede surgir información sensible que no debe ser accesible a todos.

En el diseño irán surgiendo varios documentos que serán de suma importancia para el desarrollo como modelos entidad relación para la base de datos, diagramas de flujo, diagramas de secuencias, entre otros. Por lo que toda esta información debe estar clasificada y publicada con una secuencia lógica y orden el cual ayude al equipo a identificar de mejor manera la información. Además, de que se debe ir definiendo el alcance, límites del desarrollo para tener delimitado y claro lo que se debe realizar por parte del equipo y que a su vez el cliente conozca mejor el producto que se le entregara.

Una vez finalizado la parte de diseño, se comienza a realizar la planificación, en la cual se mencionó anteriormente es muy importante apoyarse en una herramienta que permita compartir la información con el resto del equipo de una manera fácil. Aquí se debe de considerar el tomar cada una de las funcionalidades comprometidas e ir las desglosando en tareas lo más atómicas posibles para que estas ayuden a poder cuantificar y conocer el porcentaje de avance. Otros factores a considerar una vez se encuentran desglosadas las actividades es saber con cuántos recursos se cuenta para el desarrollo de las funcionalidades ya que este punto puede ser un factor determinante a la hora de la estimación del tiempo para la realización de cada una de las tareas.

Básicamente la primera fase del proceso *DevOps* involucra el analizar, diseñar y estimar el requerimiento para poderlo poner en marcha.

1.3.2. Creación

Ahora que ya ha definido el producto que se desea elaborar, como lo hará y cuánto tiempo tomará, es necesario empezar a crear lo serán tanto las bases de la aplicación y del ciclo de *DevOps*. Para ello se empleará la primera iteración si se habla de una metodología como *Scrum* o bien lo equivalente en alguna otra metodología o *framework* de desarrollo ágil en el cual se debe crear las fuentes iniciales para el proyecto, las cuales vendrían a ser los servicios rest para el *backend* y *frontend*.

Aquí se inicia con la parte de codificación de la aplicación, en la cual el desarrollo avanza en pequeños entregables. Al mismo tiempo se va definiendo las pruebas que se deben realizar al entregable para que este cumpla con los requerimientos funcionales y de calidad establecidos en la primera fase.

Este parte del ciclo de vida de *DevOps* siempre se sabe que representa la creación de una nueva funcionalidad, la solución a un defecto del código o una mejora en algún proceso. Se debe tener en cuenta que también aquí se puede ir agregando nuevas funciones que se necesiten que realice nuestro *pipeline* a la hora de ejecutar el proceso de compilado, pruebas, despliegue, entre otros.

A manera de resumir el objetivo de esta etapa del ciclo de *DevOps* es la creación de nuevo código o modificación del existente y a su vez desarrollar las pruebas necesarias al código para garantizar que cumpla con los requerimientos de funcionalidad y estándares de desarrollo.

1.3.3. Construcción

El objetivo de esta fase es la de poder tomar todas las fuentes integradas de la aplicación desarrolladas en la fase anterior y con base en ello poder construir los artefactos de los cuales está compuesta la aplicación. Esta fase es una parte crítica del proceso en la que se puede validar si al momento de realizar la integración de las diferentes fuentes de códigos se integraron de manera adecuada.

Si se habla de la parte que le corresponde al *pipeline* se puede decir que es una parte que puede cambiar dependiendo del proyecto, ya que no todos los proyectos utilizan los mismos lenguajes y las misma tecnológicas, aunque su esencia se mantiene que es la de generar los artefactos que necesita la aplicación para que funcione.

Este proceso cíclico permite de manera automática ir generando artefactos de las diferentes versiones de nuestro código y tenerlo disponible si se desea. Ya que por cada integración a la rama de develop se crea un nuevo *pipeline* que ejecuta el proceso de compilación para generar los nuevos artefactos con el código existentes y el nuevo.

1.3.4. Calidad

Cuando se habla de calidad en el *software* se habla de muchos temas sin embargo cuando se habla de la calidad en las fases del *DevOps* se habla específicamente de la ejecución de las pruebas que se diseñaron en las fases anteriores y que ahora se realizan para validar ciertas funcionalidades mínimas que se consideran deben de ser aprobadas en la ejecución de las pruebas para

garantizar que el *software* a liberar tiene cierto nivel de certeza de que va a funcionar en su despliegue.

En esta parte de ciclo de *DevOps* el *pipeline* ejecuta un *set* de pruebas que se diseñan y programan. En estas pruebas se puede garantizar que el nuevo desarrollo no afecto las funcionalidades ya existentes, además de que da un margen de certeza de que el código se encuentra probado y por ende los errores a encontrar pueden ser menores.

Dependiendo del tipo de aplicación y de los estándares que se definen con el cliente para el control de calidad se pueden ver en la tarea de tener que desarrollar pruebas unitarias para los servicios *rest* en el *backend*, pruebas automatizadas para el *frontend* o incluso pruebas de integración en la cual se ejecuten un ciclo completo de un proceso de la aplicación. Pero todo esto depende de las necesidades y exigencias del cliente.

Lo que se debe tener en cuenta es que al final en esta parte del proceso lo que corresponde es ejecutar todas las pruebas que se hayan desarrollado para la aplicación con el fin que estas sean satisfactorias y que con ello se garantice la confianza del código y a su vez generar reportes mediante herramientas como *sonarqube* en el cual se pueda ir midiendo las diferentes vulnerabilidades y niveles de cobertura de código que se tiene presente en el desarrollo (*Sonarqube*, 2022).

1.3.5. Despliegue

El despliegue es uno de los grandes objetivos de proceso de *DevOps*, ya que se pretende el facilitar el paso de un *software* que se encuentra en desarrollo a un ambiente de pruebas o de producción.

En el pasado esta parte como muchas otras del ciclo de *DevOps* se realizaban de manera manual desde la construcción de los artefactos hasta la colocación de estos en los ambientes, sin embargo, ahora se poseen herramientas que permiten automatizar este proceso mediante el uso de conexiones *ssh* hacia los servidores en los cuales realizan una transferencia de los artefactos creados y a su vez también ejecutar comandos remotos que permitan poder poner a funcionar el aplicativo.

El utilizar estas herramientas favorece el proceso de automatización y permite disminuir el factor de error humano que se involucraría si la tarea fuera realizada de manera manual.

El despliegue de la o las aplicaciones depende de cómo estas están construidas y que se requiera para que se puedan ejecutar. Al tener una aplicación que necesita de un servidor de aplicación como un *weblogic* o puede ser una aplicación basada en *Java Spring* que la cual podemos tener un *apache tomcat* embebido. Como se observa el despliegue de una aplicación puede ser muy diferente entre una aplicación y otro, pero en principio lo básico consiste en acceder al servidor destino, posicionar los artefactos en la carpeta destino y ejecutar los artefactos.

1.3.6. Verificación y Liberación

Una vez los artefactos fueron colocados en el destino designado y luego estos artefactos son puestos en marcha corresponde el validar que la aplicación tenga un comportamiento adecuado con las nuevas funcionalidades implementadas.

Para poder agregar esta verificación al ciclo de *DevOps* se deben desarrollar pruebas automáticas integrales. Con integrales o también conocidas como *E2E (End to End)*, estas pruebas ayudan a verificar de inicio al fin los flujos principales de la aplicación y que son parte de la lógica del negocio.

El apoyarse de herramientas como *SoapUI* en la cual se diseña y automatizan estas pruebas para ir realizando una serie de peticiones a los servicios de la aplicación para luego se capturen las respuestas y compararla con el resultado que se considere adecuado.

Cabe mencionar que para estas pruebas se debe definir los datos que serán enviados a los servicios a probar y las respuestas que se obtendrán para cada uno de los casos funcionales.

1.3.7. Monitoreo

La última parte de este ciclo de *DevOps* es la de monitoreo y en ella se centrarán en la verificación de los ambientes y aplicaciones. Se debe tener el control de cómo se están comportando los ambientes y servidores con el aplicativo para conocer qué tantos recursos están siendo consumidos.

Es importante tener un control de cómo están siendo utilizados los recursos disponibles ya que esto puede brindar parámetros de que tan eficiente es la aplicación o bien si los recursos de los que se disponen no son los suficientes para poder tener la aplicación corriendo de manera óptima.

El conocer este tipo de parámetros es muy útil en un ambiente *on premise* o también llamado un servidor local ya que, si en algún punto la aplicación llega a estar bajo una demanda alta el consumo de *cpu*, memoria *ram* o de disco duro

puede llegar a puntos críticos en los cuales podría ocasionar incluso un fallo en los servidores que a su vez podría provocar que estos se apaguen.

En un ambiente *cloud* se tiene la ventaja que la auto escalabilidad tanto horizontal como vertical ya que los servicios *cloud* en la actualidad cuentan con sistemas que permiten configurar los ambientes para que bajo cierta demanda o cierto nivel de estrés este pueda crecer en memoria *ram*, almacenamiento o *cpu*, lo cual en un ambiente local no es tan factible ya que requeriría el apagar los servidores y por ende dejar la aplicación fuera de línea en lo que se realiza este proceso manual, además que también implica gastos en términos de *hardware* nuevos para suplir esta necesidades y que pueda que este no sea utilizado en su totalidad en la mayoría del tiempo.

Existen herramientas como *Kibana* que permiten el tener una implementación de este proceso mucho más fácil y que permite tener esta información en tiempo real para poder analizar los picos de tráfico y tomar decisiones adecuadas en base a los datos recopilados.

1.4. Metodologías ágiles

Las metodologías ágiles son aquellas que permiten adaptar la forma en la que se está trabajando un proyecto, consiguiendo con esto flexibilidad y una pronta respuesta a cambios y solicitudes que pueden surgir por parte del cliente.

Utilizando metodologías ágiles se gana flexibilidad y autonomía en la gestión de los proyectos ya que se busca tener equipos multidisciplinarios los cuales sean capaces de irse gestionando de manera autónoma e ir distribuyendo el trabajo acorde a la disponibilidad y habilidades de cada uno.

1.4.1. *Scrum*

Es conocido por su proceso incremental el cual se basa en pequeños entregables en periodos no mayores a 3 semanas que permite el ir desarrollando entregables funcionales en un corto tiempo, a su vez el cliente no debe esperar a que el producto final esté terminado para ir conociendo parte funcionales de la aplicación.

Es un marco de trabajo utilizado para gestionar proyectos complejos en el cual se puede emplear varias técnicas o procesos (Schwaber, Sutherland, 2019). Este marco de trabajo de *scrum* se basa en el empirismo, en donde se considera que el conocimiento nace de la experiencia y la toma de decisiones con base al conocimiento propio de cada miembro del equipo.

Scrum posee una serie de roles y artefactos que permiten poder llevar una secuencia de cómo trabajar con este marco de trabajo. Entre los roles que manejamos en *scrum* está:

- *Product Owner*. Es la persona dueña del producto final y quien nos va indicando la prioridad de las user stories a trabajar en el *Product Backlog*.
- *Scrum Master*. Quien es la persona encargada de verificar que se lleve de manera adecuada el marco de trabajo de *scrum* además de ayudar al equipo de desarrollo a quitar obstáculos que no les permitan avanzar en las tareas.
- Equipo de desarrollo: Son los encargados del desarrollo y pruebas de las *user story*.

Entre los artefactos del marco de trabajo de *scrum* tenemos el ya mencionado *product backlog* que es un listado de tareas de usuarios a desarrollar, el *sprint backlog* es la lista de tareas que vamos a desarrollar en el *sprint* actual que estamos ejecutando.

Además de los artefactos se tienen los eventos de *scrum* entre los cuales está el *sprint planning* el cual consiste en una reunión no mayor a 4 horas para un periodo de *sprint* de 3 semanas en el cual se define cuáles serán las user story del *product backlog* que se trabajara. El *sprint* el evento que puede llegar a ser entre un periodo de 1 a 5 semanas dependiendo la cantidad de historias a desarrollar y es donde el equipo de desarrollo se encarga de poner en marcha el *sprint backlog* y las historias de usuarios contenidas en el.

1.5. Herramientas para la gestión de proyectos

Para el manejo de los proyectos existen múltiples herramientas en el mercado algunas de pago otras de open source sin embargo se deben centrar en un *toolset* de herramientas que permitan trabajar de manera colaborativa y en las cuales se puede interactuar entre ellas para poder brindar una mejor experiencia de usuario y a su vez tener un mejor control de la información y gestión del proyecto.

1.5.1. *Jira*

Jira es una Plataforma en línea propiedad de *Atlassian*. *Jira* tiene una serie de características que permiten gestionar proyectos e ir creando tareas o incidencias de manera muy fácil e intuitiva (Arroyave, 2021).

Jira cuenta con una serie de integraciones que permiten una mejor fluidez en el desarrollo de las tareas del equipo, se cuenta con gestión para la integración de los conocimientos en la cual se puede centralizar toda la información perteneciente a un proyecto en la sección confluence la cual es parte de *set* de herramientas, también se cuenta con hojas de rutas, *plans* o la incorporación del *plugin* de *Jira Advanced RoadMAP* en los cuales se puede agrupar una serie de tareas o incidencias y colocarles fechas de cuando se iniciaran y cuando se pretenden terminar. Esto ayuda a tener una mejor trazabilidad y visualización de los avances conforme pasa el tiempo, además de conocer el responsable de cada una de las actividades.

Jira permite el poder crear *dashboards* muy variados y tableros en los cuales se puede llevar un control de en qué estado se encuentran cada una de las actividades, el tablero que brinda *Jira* brinda la opción de iniciar a trabajar de una manera rápida con los *workflow* que ya tiene predefinidos o bien si por el giro del negocio o de la aplicación se necesita de un *workflow* mucho más a la medida la herramienta permite realizar cambios conforme las necesidades.

Para su mejor manejo cada vez que necesitamos crear una serie de tareas o incidencias, *jira* maneja una jerarquía en la que tenemos los siguientes ítems:

- Épica: Es la de mayor rango y se sabe que su contenido representa el desarrollo de una funcionalidad por completo.

- Historia o Incidencia: Representa un requisito el cual se detalla desde el punto de vista del usuario final.
- Tarea: Es el detalle de las actividades a realizar para poder tener el producto final que se detalla en la historia de usuario o incidencia.

Los *workflow* son una herramienta de la cual se pueden apoyar para poder definir un o una serie de flujos de trabajo para nuestras historias, *bug*, entre otras. Para cada flujo de trabajo se puede definir lo que son estados y transiciones:

- Estados: Permiten representar como se encuentra en la actualidad la historia o *bug*.
- Transiciones: Ayudan a definir hacia qué nuevo estado se puede transitar en base al resultado del estado actual, pudiendo definir una transición hacia un estado cuando el resultado sea éxito o bien hacia un estado diferente cuando este sea fallido como lo podría ser en el caso de un *bug*.

El objetivo de un *workflow* es ayudar a conocer en todo momento cual es el estado del desarrollo desde su estado inicial hasta llegar al estado final.

1.5.2. Confluence

Es el apartado perteneciente a *Atlassian* en el cual se puede crear lo que ellos denominan Espacios. Que son los apartados que uno puede crear para documentar toda la información necesaria del proyecto.

En *confluence* se puede ir creando en base a las diversas plantillas que dispone una serie de documentos, *blog*, artículos de cómo resolver algún problema, minutas, entre otros. Al tener una amplia variedad de plantillas se facilita el documentar el proceso de la aplicación además cuenta con diversas

plugins que permite tomar información de jira como tableros o hojas de rutas e irlos agregando en los documentos para poder llevar un control del avance.

Parte de la importancia de utilizar esta herramienta es que permite centralizar toda la información de un proyecto en un solo lugar y a la vez da las facilidades para poder restringir o permitir el acceso a cierta información en cada apartado del *confluence*.

Se puede dejar plasmados desde los casos de uso desarrollados para las aplicaciones hasta los diagramas de actividades, modelos entidad relación, diagrama de la infraestructura, entre otros. Incluso en algún caso puede ser fuente de consulta para otros equipos perteneciente a la institución o para fuentes externas.

1.6. Herramientas para la gestión de código

Las herramientas para la gestión de código permiten el tener a varios desarrolladores trabajando sobre un mismo proyecto a la vez, sin que tengan que lidiar con el problema de saber quién de ellos posee la última versión del código y también ayudan a evitar el problema de tener que actualizar el código de manera manual cada vez que alguien realiza un nuevo cambio.

Lo importante de utilizar estas herramientas es que permite gestionar el código de una manera más fácil, organizada y centralizada. Por lo que no se tienen que preocupar de quien tiene la última versión con los cambios actuales ya que para ello existe un repositorio central en el cual todos se pueden conectar y bajar la última versión y trabajar sobre ella.

1.6.1. **GitLab**

Es un servicio *web* basado en git el cual permite gestionar las versiones de código fuente de manera colaborativa. Entre otras funcionalidades que ofrece la plataforma contamos con el manejo de wikis y también la creación y seguimiento de incidencias.

Inicialmente *GitLab* era utilizando como gestor de repositorios inicialmente sin embargo con el paso del tiempo la herramienta ha ido evolucionando lo cual ha permitido el tener herramientas que complementan sus funcionalidades y han hecho ya no solo un gestor de código sino una herramienta para la implementación de *DevOps*.

Entre las diversas herramientas que posee *GitLab* está lo que es *GitLab CI*. Por qué utilizar esta herramienta por sobre otras existente y la razón es sencilla es debido a que en una sola herramienta se puede gestionar el código y a su vez crear un *pipeline* completo. Para poder crear un *pipeline* en *GitLab* utiliza una herramienta conocida como *GitLab Runners*, la cual es el vínculo entre *GitLab* hacia un servidor en la cual se pueden correr las instrucciones de *pipeline* en base al tipo de *runner* configurado siendo posibles *Shell*, *Docker*, entre otro.

Para la construcción de nuestro *pipeline* necesitamos de la creación del archivo `.gitlab-ci.yml` el cual contendrá la lógica necesaria para poder ejecutar cada uno de los *steps* que consideremos necesarios implementar en el *pipeline*. Dentro del archivo nos encontraremos con las siguientes secciones en las que está dividido el archivo:

- *Image*: Permite indicar las librerías y el sistema operativo en el que queremos basar el *runner*, esta aplica para los *runners* de tipo *Docker* ya que ayuda a definir la imagen de *Docker* que se utilizara en ciertas tareas.
- *Stage*: Permite predefinir una serie de comandos a ejecutar para la configuración. Esto implica que los *Jobs* o trabajo se trabajaran con estos comandos
- *Before Script*: Esta sección permite el lanzar uno o varios comandos antes que se inicie los *Jobs* de los *stages*.
- *After Script*: Esta sección permite ejecutar comandos que se ejecutaran una vez terminados todos los *jobs*.
- *Script*: Es un *script* de un *shell* el cual se ejecutará a través de *runner* y en el cual se puede colocar las acciones que determinaran para el *job* en el que se encuentra declarado.
- *Tags*: Esta etiqueta se utiliza en función de si se tiene diversos *runners* configurados cada uno con un propósito en particular con esta etiqueta se puede definir el *runner* que se desea que ejecute el *job*.

1.7. Servicios en la nube

Los servicios en la nube o *Cloud* han ido teniendo un mayor auge debido a las necesidades de muchas empresas que buscan el poder aumentar la disponibilidad de sus aplicaciones, pero al mismo tiempo reducir sus costos.

En la actualidad existen empresas que han logrado madurar sus tecnologías en la nube lo cual permite el poder hacer una migración completa de una aplicación *On Premise* a una en un ambiente *Cloud*. Entre las empresas líderes en brindar estas tecnologías e infraestructura tenemos *Amazon* con *AWS*, *Google* con *GCP*, *Microsoft* con *Azure*, entre otros.

1.7.1. Proveedores de servicios en la nube

En la actualidad existen varios proveedores de servicios en la nube que cuenta con una gama muy amplia de servicios e infraestructura para poder implementar nuestras soluciones en la nube. Entre los proveedores más conocidos se tiene a los ya mencionados *Amazon Web Services*, *Azure* y *Google Cloud Platform* (Bala, 2021).

Figura 3. Cuadrante Mágico de *Gartner* 2021



Fuente: Google (2021). *Gartner*. Consultado el 10 de septiembre de 2021.
Obtenido de <https://cloud.google.com/gartner-cloud-infrastructure-as-a-service/?hl=ES>.

Como se puede observar en la figura 3 se tiene el Cuadrante Mágico de Gartner, en él se observa las plataformas que brindan servicios *Cloud* en donde se puede visualizar que *Amazon Web Services* lidera el mercado y lleva una ventaja significativa con respecto a sus competidores más cercanos, en el

segundo puestos Microsoft con su nube de Azure y en el tercer puesto se tiene a Google con Google Cloud Platform. Mucha de la ventaja de Amazon tiene sobre sus competidores es por dos razones, la primera es porque ellos iniciaron a desarrollar sus servicios e infraestructura con varios años de anticipación debido a que fue en un inicio una necesidad que surgió para la empresa de Amazon, la segunda es la gran cantidad de servicios que brinda en diversas áreas de la informática que abarca desde la virtualización de máquinas, servicios *serverless*, manejo de *IOT*, *Big Data*, entre otros.

Pero tampoco se debe dejar de lado a los competidores que sin duda alguna brindan también una serie de servicios que realizan cosas muy similares. Se puede decir que las diversas plataformas que existen hoy en día y brinda el servicio de la nube posee muchos servicios muy similares que puede variar en nombre y en algunas configuraciones entre una plataforma y otra, pero que en esencia realizan los mismo. Por lo que muchas veces el preferir una sobre la otra puede ser por temas de practicidad o de gusto.

Como se observa que los proveedores tienen muchas similitudes en cuanto a servicios que brinda, también se encontraran similitudes en lo que son conocidos como regiones de disponibilidad que es donde se crearan servidores pues uno decide en qué región crear o solicitar los servicios y con base a esa decisión así pueden ser los servicios disponibles, la latencia que se puede llegar a experimentar o los costos que se pueden aplicar a los servicios que se consuman. Ya que los cobros por servicios también son similares ya que en resumen se puede decir que en un servicio pueden cobrar por el nivel de tráfico entrante o saliente, el consumo de recursos en unidades de tiempo o una tasa fija por tiempo de utilización.

1.7.2. **Software Como Servicio (SaaS)**

El termino *SaaS* o *software* como servicio hace referencia al *cloud computing* la cual consiste en ofrecer a los usuarios una solución en la nube con todo su infraestructura y plataformas. Este tipo de soluciones puede ser útil tanto para empresas de gran envergadura como pequeñas empresas ya que se desligan de temas de mantenimiento de servidores, actualizaciones de sistemas operativos o temas de redes, sino se enfocan en lo importante que es el uso de la herramienta y que esta de solución a la problemática.

Los proveedores de servicios en la nube tienen este modelo de negocio implementado en muchas de las soluciones que ofrecen pues es parte de las ventajas que se le ofrecen a los clientes. Al no tener que preocuparse por temas de mantenimiento, seguridad e infraestructura de un lugar físico los clientes empiezan a ver el retorno de inversión que le puede generar el tener un servicio en la nube versus uno que se encuentra alojado en un servidor propio.

En su gran mayoría, las aplicaciones desarrolladas bajo el modelo de *SaaS* utilizan un modelo de suscripción para poder adquirir el *software*. A diferencia del licenciamiento normal de un sistema operativo por ejemplo donde se compra una licencia de por vida este modelo nos permite asociar el *software* a una cuenta por un periodo de tiempo que pueden ir de un mes la suscripción hasta tiempos más largos como un año. Todo esto depende de cómo el proveedor maneje este tipo de suscripciones.

1.7.3. Infraestructura como Servicio (*IaaS*)

Que es *IaaS* pues como su nombre lo indica es tener un servicio en el cual yo como proveedor poseo una serie de servidores y me encargo de la infraestructura, mantenimiento de los servidores y las redes a través de internet, mientras que el usuario puede acceder a ellos mediante mi plataforma o mediante una conexión ssh hacia el servidor.

El *IaaS* es una de las tres características principales de la computación en la nube, este es uno de los sectores en mayor crecimiento debido a las facilidades y ventajas que obtiene los clientes al contratar este tipo de servicios ya que les pueden llegar a representar muchas ventajas a la hora de querer implementar un sistema.

También entre las ventajas que se tiene a la hora de utilizar este método como parte de la solución para nuestras aplicaciones es la escalabilidad vertical ya que podemos ampliar nuestras capacidades de almacenamiento, procesamiento y memoria de manera casi automática o bien decrecer en el uso de estos, si en algún momento el tener tanto recurso para una aplicación solo representa un desperdicio de recursos y de dinero.

1.7.4. Plataforma Como Servicio (*PaaS*)

Al utilizar *PaaS* se pueden desligar de los recursos necesarios para la aplicación y nos centramos más en la misma aplicación. Esto es debido a que el *PaaS* utiliza a él *IaaS* para gestionar la aplicación y provisionar los recursos necesarios para que la aplicación funcione bien con los recursos necesarios.

Existen plataformas como *Google App Engine* que permite el poder desplegar aplicaciones *web* de forma gratuita bajo la infraestructura de *Google*, en ella el objetivo es poder tener una página *web* y no tener que preocuparse por la infraestructura necesaria para que esta funcione de manera adecuada ya que la misma plataforma se encarga de gestionar el aumentar las capacidades de *cpu*, memoria *ram* o disco duro si llegaran a ser requeridos (Ravoof, 2021).

Qué diferencia se tienen entre el *IaaS* y el *PaaS* pues se tiene que la Infraestructura como Servicio solo se está utilizando un servidor virtual con sus recursos y uno se encarga de la administración de estos, mientras que si utilizamos *PaaS* se tiene una plataforma en la cual se puede configurar la aplicación y ella ya se encarga de suministrar los recursos necesario mediante *IaaS*.

1.7.5. Servicios en la nube para *DevOps*

Como se observa lo que son *SaaS*, *IaaS* y *PaaS* tienen mucho que ver con lo que conocemos de *Cloud Computing*. Debido a esto las metodologías ágiles tomaran mucha fuerza y con ello el proceso de *DevOps* y es justa mente en estas plataformas es que se encuentran herramientas que ayudan a implementar la automatización del proceso de desarrollo, mediante la implementación de servicios propios del proveedor de servicios en la nube o bien puede que la opción de interactuar con una aplicación externa que también puede ayudar.

Para ir entrando en materia el hablar de servicios que brinda tanto GCP como AWS los cuales tienen herramientas con funcionalidades similares para la implementación de un proceso de *DevOps* utilizando herramientas propias de los proveedores, tenemos el caso de *Aws CodePipeline* y *Google Cloud Build* las

cuales son servicios que brindan y en los cuales se puede realizar compilación, pruebas de manera fácil y confiable, básicamente es la parte en la que se puede implementar un *pipeline* para la ejecución automática a la hora de haber nuevos cambios en el repositorio (Aws, 2020).

Cada uno de los servicios que ofrecen los proveedores tiene sus particularidades y características que pueden hacer decidir hacia uno u el otro, en *Aws CodePipeline* se tiene las características que se crear el modelo de flujo de trabajo o también conocido como *step* y *Jobs* en el *pipeline*, tiene elementos prediseñados que pueden ser útiles y ya no es necesario el crearlo por cuenta propia y también permite el realizar el despliegue en múltiples ambientes de AWS. Por otro lado, si hablamos de *Google Cloud Buil* podemos hablar de que permite decidir que se desea de nuestros artefactos y que se pueden tener tiempos extremadamente cortos en la ejecución y construcción de los artefactos para el despliegue en los ambientes.

Como se observar el elegir una herramienta o proveedor de servicios sobre el otro es más un tema de que se necesita tener disponible para el proyecto que el elegir uno por popularidad o algún tema trivial ya que ambas plataformas ofrecen altos estándares.

1.8. React

React es una librería de *JavaScript* según la página oficial. Es un lenguaje de programación creado Jordan Walke en *Facebook* (*React*, 2021). Basado en componentes que tienen la finalidad de hacer del desarrollo *web* más rápido a manera que el renderizado sea más eficiente en los navegadores.

1.8.1. Origen de *React*

El lenguaje tenía como nombre inicialmente *FaxJs* ya que su creador fue influenciado por otro lenguaje de llamado *XHP* el cual implementaba *PHP* y *Hack* con una sintaxis de *XML* para poder crear elementos *HTML* personalizados y reutilizables.

En el año del 2013 se lanza la primera versión y es implementada en el *timeline* de *Facebook* el cual buscaba el poder cargar más contenido en su página de una manera más rápida. Posteriormente el código fue liberado de manera *Open Source* lo cual permitió su crecimiento y el tener una comunidad de desarrolladores libres detrás de ciertas mejoras

Parte del éxito de la librería y de su popularidad es debido a su velocidad y a su forma de manejar el área de componentes *web* o *DOM*, *React* trabaja con un *Dom* virtual el cual le permite tener un árbol de componentes y al momento que necesita realizar un cambio en la interfaz *web* como mostrar un mensaje u ocultar algún elemento *React* no actualiza todo el *Dom* lo que implicaría volver a cargar toda la página sino que el actualiza el nodo específico del componente que sufrió el cambio con lo cual se puede observar que la página no necesita cargarse de nuevo y realizar una serie de peticiones *rest* nuevamente para poder visualizarse sino que se actualiza la parte afectada únicamente.

La popularidad de la librería llegó al punto que posteriormente se anunció una nueva versión que sería conocida como *React Native* la cual seguía siendo basada en componente utilizando *JavaScript*, pero con la diferencia que esta sería enfocada al desarrollo móvil híbrido de aplicaciones ya que permite crear aplicaciones *JavaScript* que son compatibles con sistemas operativos *IOS* y *Android*.

1.8.2. Desarrollo de aplicaciones web basadas en componentes

Las aplicaciones *web* tradicionales se encuentran basadas en elementos visuales que es la parte que corresponde a las plantillas de *HTML* y *CSS*, mientras que la lógica iba de la mano de un archivo *Php* o *JavaScript*. Estas páginas *web* en sus inicios eran páginas estáticas y con estáticas nos referimos a páginas que no sufrían cambios que eran publicadas de una forma con ciertos elementos y que estos no iban a cambiar, mientras que las dinámicas pues pueden ir cambiando en número de elementos y acceso.

Gracias al crecimiento de las tecnologías de redes y computo fueron surgiendo nuevas necesidades que demandas por parte de los clientes que ya no se conformaban con una página estática que siempre mostraba lo mismo sino necesitaban ya página que fueran más amigables con el usuario en sus interacciones y que estas fueran mucho más fáciles de implementar, diseñar, codificar, a su vez también que estas no pusieran toda la carga del lado del navegador sino del servidor. Todos estos factores a favor permitieron el surgimiento de la programación de componentes o basada en componentes.

La programación basada en componentes pretende el poder desarrollar páginas *web* que sea mucho más rápidas y eficientes con respecto al renderizado de los elementos de las páginas ya que considera que una página *web* está conformada por una serie de elementos o componentes los cuales pueden interactuar los unos con los otros o bien pueden ser una parte aislada de la página que no afecta al resto.

Para crear páginas *web* basadas en componentes se utilizar la librería de *ReactJs* y los *frameworks* de *Angular* y *Vue* los cuales tiene su forma particular y

única para crear componentes pero que tiene la misma meta que es crear páginas *web* basadas en componentes en las cuales el *Dom* se maneje de una manera eficiente para el renderizado de los componentes en los navegadores *web*.

2. COMPARATIVA DE HERRAMIENTAS EN LA IMPLEMENTACIÓN DE *DEVOPS*

2.1. Herramientas de planeación y diseño

El ciclo de vida del proceso *DevOps* la fase de planeación y diseño es la que inicia la fase de análisis y de estimación de las futuras funcionalidades que tenemos planeado desarrollar en un tiempo determinado.

Para ello se apoya en herramientas que pueden ayudar a realizar este proceso de una manera más fácil y simple, por lo que se ira hablando acerca de las herramientas competentes para esta fase del proceso y cuáles son los puntos a favor y en contra que se tienen que considerar a la hora de inclinarnos hacia el utilizar una u otra herramienta.

2.1.1. Comparativa de *Jira* vs *Trello*

Para gestionar un proyecto se necesita de una herramienta en la cual se pueda gestionar las tareas del día a día y saber quiénes en el equipo están llevando a cabo estas tareas y el estado en el que se encuentran. Para ello existen estas dos herramientas en la cuales se puede crear tableros los cuales permiten el conocer el estado de las actividades y crear un flujo de trabajo de acorde a las necesidades.

Tabla I. **Comparativa *Jira* vs *Trello***

Atributo		<i>Jira</i>	<i>Trello</i>	Comentario
Cantidad usuario	de	10 usuarios máximo	usuarios ilimitados	Si el número de integrantes del equipo es alto y no se desea invertir en una aplicación para llevar el control de las tareas <i>Trello</i> sería la opción ya que se tiene la opción de usuarios ilimitados a diferencia de <i>Jira</i> .
Cantidad Tablero	de	Ilimitados	10 tableros máximo	Si por el contrario el equipo es pequeño, pero con varios proyectos asignados entonces se puede optar por <i>Jira</i> ya que se puede crear varios tableros en su versión gratuita
Integración terceros	con	Disponibles para varias aplicaciones de terceros	Disponibles para varias aplicaciones de terceros	En cuanto a la integración de la herramienta con app de terceros se sabe que ambas herramientas soportan esta opción sin embargo la

Continuación de la tabla I.

				diferencia radica en que <i>Jira</i> tiene integración con una gama más amplia de app.
Reportes	Cuenta con una serie de diagramas y herramientas para generar reportes y <i>Dashboard</i> .	En su versión gratuita no posee esta opción.		Si se desea tener datos porcentuales de avances y gráficas que indiquen el tiempo y avance sin duda <i>Jira</i> es la opción sin dudarlo ya que <i>Trello</i> no posee dicha opción
Agrupación de tareas y tipificación	Existen lo que son las Épicas, Historias, entre otras.	Existen la tarea o tarjetas, pero no existen más tipificaciones		Si se desea tener una jerarquía, orden y una agrupación de las tareas a realizar para el control en el proyecto las opciones que brinda <i>Jira</i> sería la mejor opción.

Fuente: elaboración propia, realizado con Microsoft Word.

Al observar cada uno de los puntos comparativos entre las herramientas se puede decir que *Trello* es una herramienta que nos permite el poder llevar el control de tareas de diversas índoles y no solo de desarrollo de *software* ya que esta herramienta cuenta con plantillas para dicha función, sin embargo, *Jira* es la opción más óptima si el equipo de desarrollo desea poder gestionar varios proyectos y generar métricas del porcentaje de avance en base a la tareas creadas así como el de poder llevar acabo *sprint* y tener *Sprint Backlog* y *Product Backlog* para cada una de nuestras iteraciones en el proyecto.

2.1.2. Comparativa Xray vs Kualitee

Se ha visto que en el proceso de *DevOps* la gestión y creación de tareas e incidencias es importante, sin embargo, al encontrarse ya en el desarrollo de las funcionalidades se debe de pensar siempre en las pruebas que se realizan al *software* que se va a entregar y para ellos existen herramientas en las cuales también se puede llevar esa parte de gestión y desarrollo de pruebas. Existen herramientas que ayudan con estas tareas para que el producto final cumple que una cierta cantidad de criterios mínimos para su correcto funcionamiento.

Tanto *Xray* como *Kualitee* permite tener un control sobre las pruebas a realizar y sobre los resultados que se obtendrán, permitiendo visualizar la información mediante *Dashboard*.

Tabla II. Comparativa Xray vs Kualitee

Atributo	Xray	Kualitee	Comentario
Gestión de casos para desarrollo de pruebas	Disponible	Disponible	Para estos se sabe que en ambas herramientas se puede crear las historias y gestionar historias para poder manejar las pruebas por lo que no hay diferencia entre este punto

Continuación de la tabla II.

Manejo de archivos y clonación de pruebas	Disponible	Disponible	Si se desea cargar archivo para las pruebas o clonar pruebas entre proyecto si se ve que se pueden reutilizar, el único que brinda esta opción es <i>Xray</i> ya que <i>Kualitee</i> no posee ninguna de estas opciones a la hora de crear el set de pruebas
Operaciones por lotes	Disponible	Disponible	Cuando se tiene un set de pruebas completo desarrollado y se desea cargar al sistema ambas herramientas permiten las opciones de (clonación, edición, eliminación)
Parametrización de casos y conjunto de pruebas	Disponible	No disponible.	Si se desea crear parametrizaciones para las pruebas solo <i>Xray</i> permite realizar esta acción ya que <i>Kualitee</i> no la tiene integrada
Flujo de trabajo personalizado	Disponible	No disponible	La funcionalidad permite crear un propio flujo de trabajo para el desarrollo de las pruebas con base a las necesidades del proyecto o de la empresa

Continuación de la tabla II.

Gráficas, métricas, informes paneles personalizados	Disponibile y	Disponibile	Sin duda una de las funcionalidades vitales en la aplicación ya que en base a ella podemos analizar el avance de las pruebas y conocer mediante gráficas y métricas la información de las incidencias reportadas
--	------------------	-------------	--

Fuente: elaboración propia, realizado con Microsoft Word.

Al observar cada una de las características con las que cuenta cada una de las herramientas se puede decir para el control de las pruebas que se realizan para una aplicación es importante que la herramienta permita el crear pruebas por lotes cargadas mediante algún archivo y el también poder clonar pruebas entre proyecto ya que puede darse el caso que en algún proyecto se puedan reutilizar pruebas lo cual ayuda en lo tiempo de desarrollo de las pruebas, además el poder observar toda esta información mediante gráficas y *Dashboard* que ayude a poder interpretar los datos y los avances de las pruebas de una mejor manera es de vital importancia.

Cabe mencionar que una consideración importante a tomar en cuenta es el que las herramientas se encuentren unificada y no se tienen que llevar procesos en paralelo con diferentes herramientas que hagan realizan trabajo doble, por lo que el utilizar *Xray* brinda esta ventaja ya que es parte de los complementos que brinda la plataforma de *Jira*.

2.1.3. Comparativa *Porfolio Jira vs Microsoft Project*

Parte importante del ciclo de *DevOps* es la planeación y ejecución del o los proyecto, pero para ello se necesita de una herramienta que permita poder visualizar las tareas e irlas organizando de tal manera que están puedan también sufrir cambios si la situación lo amerita. También el poder asignar tareas a los miembros del equipo y poder medir el tiempo que les llevara el realizarlas (Sepúlveda, 2019).

Por lo cual se realizó un análisis entre 2 herramientas que permiten realizar estas tareas con el fin de analizar un poco más afondo las similitudes o diferencias que estas pueden tener, además de exponer funcionalidades que pueden que sean de utilidad al momento de querer llevar el plan de trabajo de una mejor manera.

Tabla III. **Comparativa *Porfolio Jira vs Microsoft Project***

Atributo	<i>Microsoft Project</i>	<i>Porfolio Jira</i>	Comentario
Acceso móvil, web y escritorio	Soporta <i>web</i> y escritorio	Disponible	Parte importante del trabajo en equipo es el acceso a las herramientas desde cualquier dispositivo móvil o computadora lo cual permite gestionar y estar siempre al tanto de las actualizaciones por lo cual <i>Jira</i> brinda esta opción lo cual permite poder estar al tanto, aunque no se esté en una computadora.

Continuación de la tabla III.

Creación automática de hoja de rutas	Se debe crear	Disponibile	Esta funcionalidad puede ayudar mucho en ahorro de tiempo ya que <i>Jira</i> maneja una jerarquía para las tareas las cuales al estar agrupadas e irle colocando una fecha de inicio a fin crea de manera automática la hoja de rutas.
Manejo de metodologías ágiles, <i>scrum</i>.	No disponible	Disponibile	Aquí se puede observar una diferencia significativa entre las 2 herramientas ya que una carece del soporte para trabajar con algún tipo de metodología ágil o <i>scrum</i> y solo se basa en la creación de tareas y asignación de fecha.
Control de permisos accesos	No disponible	Disponibile	El nivel de acceso muchas veces es un tema para tener en cuenta ya que no siempre se desea que toda la información sea visible para todos, <i>Jira</i> nos brinda esta opción

Continuación de la tabla III.

				de poder restringir el acceso a ciertos usuarios en cambio Microsoft Project no brinda esta facilidad.
Diagramas <i>Burn Down</i>	de	Disponible	No disponible	Esta opción permite el tener un diagrama específico para el trabajo pendiente lo cual puede ser útil, sin embargo, <i>Jira</i> , aunque no tiene una funcionalidad específica para esta opción puede realizarlo a través de sus filtros.
Integración plataformas terceros	con de	Solo con algunas herramientas Microsoft	Disponible	Aquí se observa una gran ventaja de <i>Jira</i> ya que la plataforma soporta integraciones con múltiples herramientas como <i>Git</i> , <i>Bitbucket</i> , <i>slack</i> , entre otras.

Fuente: elaboración propia, realizado con Microsoft Word.

Con base en los datos comparativos que se observan se concluye que si necesita llevar el control de las actividades de un proyecto y sobre de ello se necesita aplicar una metodología ágil en el control del proyecto el *Porfolio* de *Jira* es la herramienta adecuada ya que permite integrar metodologías ágiles o *scrum* en el desarrollo del proyecto además que al cargar historias de usuarios de manera masiva *jira* crea ya en automático una hoja de ruta en la cuales se puede ir midiendo el tiempo de cada una de las tareas y tener una perspectiva mejor de

cómo se encuentra el proyecto en términos de tiempo consumido y disponible. Mientras que Microsoft Project permite llevar la secuencia de un proyecto el cual no tenga una mayor complejidad y que solo se necesite el llevar una secuencialidad lineal de los tiempos y las tareas.

2.1.4. Comparativa *Confluence* vs *AnswerHub*

Durante la fase de planeación y diseño se debe tener presente las tareas y los tiempos en el proyecto, sin embargo, hay algo más que se debe tener en cuenta y es el documentar la información del proyecto de manera adecuada por lo cual las herramientas para la gestión del conocimiento con las cuales se puede llevar un mejor control de la información concerniente a los proyectos que se estén gestionando.

Parte importante de un proyecto llevado de una manera correcta es el tener bien documentado todos los procesos y que estos puedan ser accesibles al equipo o personas involucradas.

Tabla IV. Comparativa *Confluence* vs *AnswerHub*

Atributo	<i>Confluence</i>	<i>AnswerHub</i>	Comentario
Disponibilidad de la aplicación en medio web, móvil y escritorio.	Disponible	Disponible en su versión web y de escritorio	Los dispositivos móviles son utilizados hoy en día con una gran frecuencia que el tener las aplicaciones en su versión móvil permite estar siempre al corriente, aunque no se esté en el computador

Continuación de la tabla IV.

				lo cual hace esto de <i>Confluence</i> en su versión móvil una opción importante a considerar debido a la disponibilidad que brinda al ser móvil también.
Notificaciones por nuevas publicaciones o menciones.	Disponibles		No disponibles	Esta opción es de ayuda y es importante tomar en cuenta ya que al haber nuevo contenido en la plataforma esta avisa al grupo de nueva información o bien si alguien es mencionado.
Control de versiones	Disponibles		No disponibles	Al gestionar la información muchas veces un documento puede tener varios cambios e ir teniendo diferentes versiones por lo cual poder versionar archivos y documentación en el <i>confluence</i> ayuda a realizar una comparativa entre las diferentes versiones.

Fuente: elaboración propia, realizado con Microsoft Word.

Con base en la comparativa de los puntos importantes de las herramientas se puede decir que *AnswerHub* es una herramienta más de preguntas y respuesta en las cual podríamos postear temas de cómo resolver problemas que

se presentaron durante el desarrollo o tutoriales, mientras que en *confluence* se pueden crear arboles de información y seccionar la información por categoría además de que en los documentos se puede utilizar otros componentes de *Marquet place* de *confluence* con el cual pueden complementar muy bien los documentos ya que se tengan, una gama muy amplia de elementos que puede ser útiles como gráficas, referencias informes, entre otros.

En *confluence* se puede encontrar una herramienta completa y con más opciones para todos los documentos ya que se puede agregar diagramas mencionar a colegas e incluso incluir tableros de *Jira* de algún proyecto en curso.

2.1.5. Conclusión de herramientas de planeación y diseño

Al ir comparando diversas herramientas que serán útil a la hora de implementar el ciclo de *DevOps* se debe indicar que se puede inclinar por el *Toolset* de herramientas de *Jira* entre las cuales se tiene *Confluence* y sus *plugins* de *Xray*, *Portfolio*, entre otras. Esto es debido a que cada una de las herramientas se pueden relacionar la una con la otra por lo cual no se necesita tener un grupo de entre 3 a 4 herramientas para esta parte del ciclo de *DevOps* que lo que produciría aumentar la dificultad en el trabajo en equipo ya que no es apropiado el estar moviendo entre una y otra para poder ver la información demás de que se debería de estar retro alimentando a cada una de estas. Sin embargo *Jira* permite el ir creando tareas que a sus vez pueden tener una jerarquía en donde están las épica, historias de usuarios o *bug* y luego las tareas, creadas estas *jira* ya permite visualizar los tablero con el estado de las tareas y también crea una hoja de rutas en la cual se puede ver los tiempos de estas tareas y todo esto monitoreado desde una solo aplicación en la cual con esta información centralizada se puede ir creando reportes y *Dashboard* para ir conociendo el avance porcentual de cada una de las tareas y quienes la están ejecutando.

Por lo cual se considera que *jira* es la mejor herramienta por su sencillez su intercomunicación y unificación de la información y porque permite interactuar con herramientas de terceros que amplia aún más el uso que le podemos dar.

2.2. Herramientas de creación

Ahora se tratan las herramientas en el proceso de creación en el ciclo de vida de *DevOps* y es aquí donde entre en juego las herramientas que permitirán crear los *pipelines* con el cual se podrá llevar a cabo una serie de *stages* para posteriormente poder poner una aplicación en un servidor para que esta quede funcional de manera automática, pues este es el objetivo que se desea alcanzar.

2.2.1. Comparativa *GitLab CI* vs *Jenkins*

Dos de las más conocidas herramientas utilizadas en los procesos de *DevOps* y se van a comparar las cualidades de cada una de estas herramientas y bajo qué circunstancias se puede utilizar una u otra. Cada una tiene su forma particular para trabajar sin embargo el objetivo de ambas es el de poder implementar un *pipeline* completo para la automatización del proceso de desarrollo de *software*.

Tabla V. Comparativa *GitLab CI* vs *Jenkins*

Atributos	<i>GitLab Ci</i>	<i>Jenkins</i>	Comentario
Ejecución de pipeline paralelo	Disponible	Disponible	Esta opción permite el poder ejecutar de manera simultánea varios procesos de despliegue y que estos no se vean afectados o necesiten encolamiento.
Necesidad de infraestructura	No necesario	Necesario	Se tiene un punto muy importante a tomar en cuenta ya que si se decide el trabajar con Jenkins se necesitará de infraestructura para poder instalar la aplicación ya que esta no cuenta con la opción de un de un servicio <i>SaaS</i> como lo es <i>GitLab</i> ya que su función de <i>DevOps</i> ya está integrada en la plataforma en la nube
Control centralizado de pipeline por proyecto	No disponible	Disponible	Si se lo que se desea es poder gestionar todos los proyectos desde un punto central entonces Jenkins es el indicado ya que en su panel se puede observar todos los proyectos y sus <i>pipelines</i> a

Continuación de la tabla V.

				diferencia de <i>GitLab</i> que solo permite ver el <i>pipeline</i> del proyecto de <i>git</i> en el que nos encontramos.
Soporte para integración con aplicaciones de terceros	Disponibile		Disponibile	Esta funcionalidad está presente en ambas herramientas y de muchas formas tanto para interactuar con alguna herramienta de un tercero como en el uso de <i>plugins</i> para agregar nuevas características y funcionalidades al <i>pipeline</i> como en el caso de <i>Jenkins</i> .
Complejidad de implementación de <i>pipeline</i>	Media		Medio – Alto	La complejidad de la implementación de un <i>pipeline</i> va a depender muchas veces de las características del proyecto sin embargo las herramientas muchas veces suman también cierta complejidad en la implementación y en el caso de <i>Jenkins</i> tenemos una complejidad un tanto más elevado debido a que para realizar algunas implementaciones

Continuación de la tabla V.

			en el <i>pipeline</i> se necesita de ir instalando <i>plugins</i> para que <i>Jenkins</i> pueda ejecutar las acciones que se necesiten
Control de ramas	Disponible	No disponible	<i>GitLab</i> permite gestionar las ramas de los proyectos desde su misma página ya que se tiene integrado todo en un mismo sitio <i>web</i> , mientras que <i>Jenkins</i> no brinda esta opción ya que es una herramienta separada de nuestro control de versiones.

Fuente: elaboración propia, realizado con Microsoft Word.

Al inicio del capítulo al comparar las características de ambas herramientas y conocer tanto sus fortalezas como debilidades se puede concluir que debido a las múltiples funcionalidades que brinda además de que permite gestionar tanto el código como el flujo de los *pipelines* de manera centralizada, *GitLab* es la mejor opción. Ya que es parte de los complementos que brinda *GitLab* por defecto, además de que no es necesario el tener un servidor local o en la nube para realizar la instalación.

2.3. Herramientas de calidad

La calidad en el *software* es de vital importancia ya que como desarrolladores permite el decir que se está entregando un *software* que cumple con una serie de criterios previamente establecido que aseguran que el software no solo funciona de una manera correcta, sino que también el utilizar la herramienta es seguro (Hernández, 2018). Por otra parte, el cliente también tiene una garantía de que el producto final que se le entrega cumple con los criterios que el estableció o a los que se establecieron.

En la actualidad la calidad del *software* se puede ir midiendo con base en diferentes pruebas y utilizando diferentes herramientas. Ahora sabiendo lo importante que es la calidad del *software* es por ello que se implementa en el ciclo de *DevOps* los *stages* de pruebas y de análisis de código, aquí es donde entran las herramientas que se analizaran que tiene como objetivo el analizar el código generado para la aplicación y verificar que este cumpla una serie de normativas para analizar estos datos e indicarnos de una manera más visual, fácil e intuitiva el nivel de seguridad y la cantidad de fallos que pueden encontrarse en nuestra aplicación.

Tabla VI. **Comparativa SonarQube vs Kiuwan**

Atributos	SonarQube	Kiuwan	Comentario
Disponibilidad en múltiples plataformas	<i>Web, windows</i>	<i>Web, windows</i>	Ambas aplicaciones cuentan con una versión para la <i>web</i> o <i>windows</i> sin embargo para dispositivos móviles u otros sistemas

Continuación de la tabla VI.

				operativos no cuentan más que acceso por medio de un navegador a la <i>web</i> . Por lo que es una carencia para ambas herramientas.
Análisis de vulnerabilidades en tiempo real	Disponible	Disponible		Se puede realizar un análisis de las vulnerabilidades en tiempo real o bien cuando se ejecute un <i>pipeline</i> y este genera la información del análisis del código
Creación de informes y estadísticas de fallos y vulnerabilidades	Disponible	Disponible		Cada una de las herramientas permite el poder visualizar mediante estadísticas o gráficas la información de las vulnerabilidades y fallos que se pueden encontrar en la aplicación
Flujo de trabajo basado en reglas	Disponible	No disponible		Esta característica permite el analizar el código basado en reglas las cuales pueden ser personalizadas, en base a las necesidades y

Continuación de la tabla VI.

	criterios que se establezcan para los proyectos.
--	--

Fuente: elaboración propia, realizado con Microsoft Word.

Al comparar las características de las herramientas para el análisis de código estático se puede decir que se elige *Sonarqube* por sobre *Kuiwan* debido a las facilidades que otorga a la hora de poder visualizar la información del análisis del código en la cual se encontraran una serie de parámetros que ayudan a conocer a gran detalle cómo se encuentra el código analizado, entre los valores que se pueden destacar son que indica la cantidad de *bug* que hay en el código y el nivel de cobertura que este tiene con respecto a las pruebas unitarias.

Además, otra gran virtud es que permite crear o incluso anular reglas para la evaluación de nuestro código lo cual lo hace una herramienta muy versátil y adaptativa.

2.4. Herramientas de despliegue

Al momento de haber superado cada una de las diferentes etapas del *pipeline* y tener ya listo los artefactos de nuestros procesos de *DevOps*, se puede realizar la entrega y despliegue de estos, sin embargo, como en cada una de las diferentes etapas de la automatización se necesita de la ayuda de alguna herramienta que ayude con este objetivo.

2.4.1. Comparativa *Ansible* vs *Fabric*

Para ello es que se hablara y comparar dos de las herramientas más conocidas y utilizadas para este fin, se habla de *Fabric* y *Ansible* dos herramientas con las cuales se puede escribir comandos *ssh* en un archivo y con ello poder administrar una serie de servidores.

Tabla VII. Comparativa *Ansible* vs *Fabric*

Atributos	<i>Fabric</i>	<i>Ansible</i>	Comentario
Curva de aprendizaje	Pequeña	Media	La curva de aprendizaje de <i>fabric</i> es menor por lo que se podrá obtener resultados en un periodo menor, sin embargo, aunque <i>ansible</i> requiere de más esfuerzo para su aprendizaje su mayor potencia hace que este sea buena opción para utilizar.
Manejo de los servidores	Disponible	Disponible	En ambas herramientas se puede hacer gestión de servidores mediante comandos <i>ssh</i>

Continuación de la tabla VII.

Auto instalación	Disponible	Disponible	Las herramientas permiten la instalación de la propia herramienta y de otras de manera automatizado debido a que se tiene acceso a servidor de manera remota mediante <i>ssh</i>
-------------------------	------------	------------	--

Fuente: elaboración propia, realizado con Microsoft Word.

Al comparar ambas herramientas que prácticamente realizan lo mismo en algunos aspectos como la administración de los recursos IT y en otro varían como en la curva de aprendizaje, sin embargo, se puede decir que la herramienta que se considera la mejor para la implementación es Ansible debido a que aunque su curva de aprendizaje es un poco más alta, la herramienta le permite tener soluciones más completas y robustas, esto va de la mano con el lenguaje de *.yaml* el cual es el que utiliza Ansible. También el que no necesite un servidor central para la administración, sino que dependiendo de la necesidad el servidor que realiza las tareas de *DevOps* puede ser el encargado de realizar las conexiones y configuraciones es un factor que ayuda en la versatilidad de la solución y por estos motivos se considera a Ansible como la mejor herramienta.

2.5. Herramientas de verificación y liberación

La última etapa del proceso y es que una vez la herramienta haya sido desplegada, aún queda una parte de verificación en la cual se de constatar que

el producto final puesto en el ambiente de desarrollo o de producción se construyó de manera adecuada y funciona correctamente. Por lo que se pueden apoyar en herramientas que permita el poder verificar esta situación.

Para apoyarse en herramientas en las cuales se pueden realizar pruebas *backend* en donde se validará el ciclo completo de los procesos que consideramos el *core* de la aplicación y que por ende deben de estar disponibles para considerar exitoso el despliegue en el ambiente.

2.5.1. Comparativa *SoapUI* vs *Postman*

Al momento de realizar la verificación de nuestras aplicaciones se debe decidir qué procesos o flujos se validarán para posterior a ello se pueda implementar la automatización de estos procesos mediante una herramienta. Aquí es donde entran en juegos las herramientas las cuales se pueden ir analizando en cuales se puede consumir un *api rest* que posteriormente dará un resultado el cual comparan para ver si la respuesta es la que se esperaba y con ello seguir realizando una serie de pasos hasta completar el proceso entero.

Tabla VIII. Comparativa *SoapUI* vs *Postman*

Atributos	<i>SoapUI</i>	<i>Postman</i>	Comentario
Colaboración entre miembros de equipo	No disponible	Disponible	<i>Postman</i> brinda la capacidad de que varios usuarios pueden trabajar en simultaneo sobre un mismo proyecto a diferencia de <i>SoapUI</i> que se crea

Continuación de la tabla VIII.

				un proyecto y si se tiene más de un colaborador cada uno de ellos van agregando sus cambios sin que el otro los conozca hasta que realice un pull de los cambios si el proyecto se encuentra en un git.
Creación de test suites, test case y test step	Disponible	No disponible		<i>SoapUI</i> permite categorizar las pruebas y agruparlas para poder crear <i>suites</i> de pruebas para poder testear una funcionalidad completa, mientras que <i>SoapUI</i> lo que permite es crear grupos para los servicios lo cual no es de utilidad si lo que deseamos es realizar pruebas automatizadas de <i>testing</i>
Informes de pruebas	Disponible	Disponible		Ambas herramientas cuentan con la opción y la diferencia radica en que <i>SoapUI</i> genera reportes en formato JSON y HTIM, mientras que <i>SoapUI</i> acepta los dos anteriores además de

Continuación de la tabla VIII.

Lenguaje Scripting	de JavaScript	Groovy	informes en consola e informes analíticos avanzados.
			El <i>scripting</i> es la forma en la que las herramientas permite agregar un nivel de programación a las pruebas en el caso de cada herramienta trabajo con su propio lenguaje por lo cual es de considerar la curva de aprendizaje si fuese necesario.

Fuente: elaboración propia, realizado con Microsoft Word.

Una vez visto los puntos a comparar entre las dos herramientas se puede decir que *SoapUI* es una herramienta especializada en temas de creación de pruebas automatizadas y que por otro lado Postman es una herramienta que se utiliza con la finalidad de testear servicios rest de aplicaciones y documentarlos mas no automatizar un proceso de pruebas.

Por lo que se puede indicar que la herramienta adecuada para la automatización de estas pruebas es *SoapUI*, además de que también cuenta con una opción en la cual se puede integrar la ejecución de estas pruebas con *Jira* y poder crear una historia en el cual se puede informar del resultado de estas pruebas de manera automática.

2.6. Lenguajes de programación orientados a componentes

Los lenguajes de programación han ido evolucionando a lo largo de los años en base a las nuevas tecnologías y necesidades que han surgido. Se inició con programación secuencial con lenguajes como C y con el tiempo surgió el paradigma de objetos y con ello la programación orientada a objetos, hoy se tiene nuevos retos que han hecho surgir nuevos lenguajes con enfoques hacia las matemáticas para temas de *Maching Learning* e *IA* o temas como programación concurrente con lenguajes como *Golang* y también se tiene los lenguajes de programación orientados a componentes como lo son Angular, View, React js los cuales su enfoque se basa en tener pequeños elementos de una página *web* que puedan ser re utilizados y que estos a su vez se puedan actualizar sin la necesidad de refrescar toda la página.

Pues los lenguajes basados en componente lo que buscar es simplificar el desarrollo de aplicaciones *web* tradicionales en las cuales la página *web* al querer que hace un cambio entre un menú u otro tener que renderizar todo de nuevo lo cual repercute en tiempo y recursos, con estos nuevos lenguajes este enfoque cambia debido al mande del *DOM* virtual es cual es un árbol jerárquico de componente donde se tiene toda la estructura de elementos *web* y lo que se hace ahora es ir a buscar la rama del árbol donde está el elemento de sufrió el cambio y cambiar esa parte si tener que actualizar o crear por completo el árbol nuevamente.

2.6.1. Comparativa *React js* vs *Angular*

Los lenguajes de programación orientados a componentes han ido tomando un gran auge en los últimos años debido a las ventajas que estos brinda

y por el respaldo de quienes están detrás de estos dos lenguajes ya que tenemos a *Facebook* detrás de *React* y a *Google* por el lado de *Angular*.

Ambos lenguajes son basados en *JavaScript*, aunque con la diferencia que *React* es considerado una librería mientras que *Angular* es un *framework*, además de que ambas son de código abierto con el apoyo de una comunidad de desarrolladores de código abierto.

2.6.1.1. Desarrollo para móviles

Si se habla de la versatilidad de los lenguajes cabe mencionar que estos dos lenguajes han trascendido de ser lenguajes de programación para páginas *web* a tener su versiones respectivas para desarrollo móvil como lo son *React Native* y *NativeScript* las cuales permiten seguir bajo el esquema de manejo de componentes pero ahora ya no hablando de un entorno *web* sino uno móvil híbrido en el cual no se necesita aprender otro lenguaje de programación móvil sino se mantiene la codificación de *JavaScript* o *TypeScript* para el desarrollo de estas aplicaciones lo cual ayuda en tiempo y costos ya que no es lo mismo hacer dos veces una aplicación para una plataforma *Android* y otra para *IOS*, ya que cada una cuenta con su lenguajes propio y sus formas de implementar ciertas funcionalidades o interacción con elementos nativos de los dispositivos móviles.

2.6.1.2. Curva de aprendizaje

La curva de aprendizaje de *react* no es muy alta esto debido a que al ser una librería para *UI* esta no implementa funciones muy complejas y no tiene inyección de dependencia, sin embargo también al no poseer una estructura de proyecto estándar puede complicar su inicio ya que es algo que puede tomar

cierto tiempo aprender definir la estructura que más se adecue a nuestras necesidades.

Por el lado de *Angular* se tiene su curva de aprendizaje comparada con la de *React* es más pesada esto debido a que se tiene que tomar en cuenta que es una *framework* más grande el cual aprender sus conceptos llevara mucho más tiempo, también se debe tomar en cuenta que en *Angular* se utilizar *TypeScript* que, aunque es muy similar a *JavaScript* no es lo mismo por lo cual también es un factor que suma tiempo y esfuerzo a la hora de empezar a desarrollar en él.

2.6.1.3. Desempeño

En el desempeño *React* lleva una ventaja sobre *Angular* y esto es debido a la incorporación de *Dom* Virtual los cuales ayuda en el renderizado de los componentes y que estos se encuentran del lado del servidor y del navegador lo cual reduce la carga de este. En cuanto a *Angular* su rendimiento se ve más afectado debido al enlace de datos bidireccional que maneja con los observadores ya que por cada enlace se crea un observador y en cada ciclo se debe verificada cada uno de los observadores lo cual hace de estos un proceso no tan eficiente en comparación a como lo maneja *React*.

2.6.1.4. Comentario

Al comparar los lenguajes de programación y observar la forma en que ambos se desempeñan, se puede indicar que, aunque ambos son orientados a componentes y que incluso posee versiones para el desarrollo móvil, cuando se trata del rendimiento *React* tiene una ventaja significativa sobre *Angular* debido a la implementación de *Dom* Virtuales los cuales favorecen en la respuesta de las peticiones que hacen el navegador *web* hacia el servidor de aplicaciones.

También otro punto de importancia es que la curva de aprendizaje es mucho menor en comparación con la de angular y si sumas a estos que *Angular* se codifica en *TypeScript* aquí tenemos otro punto a favor de *React* el cual sin duda nos permite el implementar de manera rápida y sencilla una aplicación y por lo cual se considera la mejor opción si lo que necesitan es desarrollar una aplicación *web* de manera pronta y con las ventajas que esta librería ofrece.

3. ESTÁNDARES DE PROCESOS

3.1. Manejo de documentación en *Confluence*

La herramienta de *confluence* se sabe es la herramienta que se estará utilizando para poder documentar todo lo correspondiente al *DevOps* y a lo que corresponde de la aplicación que se estará automatizando.

Se creará una estructura en la cual se pueda organizar la documentación de una forma en la cual una persona con conocimiento del proyecto o alguien nuevo se pueda orientar de una manera intuitiva de donde puede encontrarse la información que necesita.

Por lo cual la estructura para el manejo de la documentación del proyecto en *confluence* se sugiere mantenga el siguiente formato:

- Configuración

En este apartado se debe documentar todo lo correspondiente a las herramientas necesarias para el desarrollo y levantado de la aplicación en un ambiente local, así también las configuraciones necesarias para que la aplicación puede funcionar de una manera adecuada.

- Solución de problemas

Se debe tomar en cuenta que es importante documentar si existiera algún caso particular que se podría suscitar en la configuración de alguna herramienta

y como resolverlo es importante, realizar esta parte de ir alimentando este apartado ya que esto también permite crecer en conocimiento a la hora solventar ciertas situaciones como equipo.

- Desarrollo

El objetivo de la sección de desarrollo es ir agregando artículos que brinde información correspondiente al ambiente de desarrollo, aquí se puede documentar información como accesos a la base de datos del ambiente o como llegar al servidor mediante ssh, también aquí se puede dejar documentado información de cómo realizar una configuración perteneciente a este ambiente.

- *DevOps*

Esta sección está diseñada para ir documentando la implementación de las herramientas que se empleara en el proceso de automatización. Para documentar de manera correcta cada herramienta debe tener su sección propia y dentro de ella tener solo artículos o temas correspondientes a la herramienta.

- Esquema del proyecto

En todo proceso de desarrollo de *software* existe una fase de diseño de la cual hablamos al inicio del proceso de *DevOps* y con la cual inicia también el ciclo de vida de una aplicación. Esta sección la dividiremos en dos partes las cuales serán:

- Análisis

La sección de análisis permitirá tener un listado de los diferentes análisis que se llevaran a cabo para dar solución a los problemas que se presenta mediante la implementación de la aplicación.

- Diseño

Para la parte del diseño tendrán todo lo correspondiente a los elementos diseñados para la aplicación, aquí se puede documentar los diferentes esquemas, mapas mentales, modelos, entre otros, que vayan surgiendo durante las fases de análisis y diseño de nuestra aplicación.

Temas como mapeos de direcciones *IP* o arquitectura del proyecto se puede ir agregando a esta sección a manera de centralizar toda esta información en una sola sección. Como punto a tomar en cuenta es que es muy recomendable el subir los archivos en los que se realicen esquemas o modelos para poder tener las fuentes disponibles, modificarlas e ir las versionando a manera de tener este control de cambios.

- Estándares

El objetivo de los estándares es definir un esquema sobre el cual trabajar de una manera uniforme por lo que en este apartado buscamos justo esto, el poder definir los estándares de cómo utilizar las herramientas o incluso estándares que se desea manejar a nivel aplicación, proceso. Lo importante es que esta información siempre quede documentada para que pueda ser consultada posteriormente por los miembros del equipo o por miembros nuevos a los cuales se necesita que conozcan cómo realizar los procesos.

- Reuniones y seguimiento

El proceso de *DevOps* es un proceso que es iterativo y por consecuencia cuenta con varias iteraciones de manera consecutiva, si a esto se suma el utilizar un marco de trabajo como *Scrum* se tendrá que periódicamente se debe realizar sesiones de seguimiento y de control de la aplicación con el cliente y equipo de desarrollo. Es por lo que en esta sección se pretende que esas reuniones de las cuales pueden salir minutas con ítems de responsabilidad para ciertos individuos se documenten en esta sección para poder tener el histórico de fechas y puntos tratados.

También aquí se puede llevar el control de las retrospectivas de *Scrum* y de los puntos a mejorar para darle un seguimiento si hemos avanzado con respecto a los puntos de mejora.

Tomando en cuenta lo mencionado en los dos párrafos anteriores la forma de trabajo que se debería de llevar el proceso de *scrum* sería inicialmente tomar lo desarrollado en el análisis y diseño de la aplicación y crear nuestro *sprint backlog* el cual se ordena en orden de prioridades, luego se procede con el *sprint planing* para definir el *sprint backlog* y ponderar las historias en base a los criterios de equipo de *scrum*, además tener presente que la reunión diaria de *sprint* conocida como *daily* se debe realizar todos los días en el mismo lugar y mismo horario para evitar complejidades y poder conocer avances o impedimentos para el avance del desarrollo

3.2. Creación de Épicas, historias de usuarios y tareas en *Jira*

Como parte de las buenas prácticas que se desea ir implementando a lo largo del proceso el clasificar y definir el uso adecuado que le dará a cada uno

de estos elementos en *Jira* es importante ya que nos permitirá el poder agrupar de una mejor manera la información para cada una de las funcionalidades que se pretenden desarrollar.

Se debe de trabajar cada uno de los siguientes ítems bajo las siguientes premisas:

- Épicas

Las épicas serán las de mayor jerarquía y en ella se agrupan una serie de historias de usuario que permitirán el tener como resultado final una funcionalidad completa. Es importante tomar en cuenta que en la épica se debe detallar lo que será el producto final entregado de manera que esta que un usuario final puede leer y comprender cuál es el objetivo de la funcionalidad desarrollada.

Entre los campos importantes que se debe completar para poder alimentar correctamente la herramienta de *Jira* y con ello generar reportes que ayuden a analizar el avance del desarrollo de manera eficiente se tienen los siguientes:

- Título
- Descripción
- Fecha Inicio
- Fecha Final
- Responsable

- Historia de usuario

Las historias de usuarios estarán agrupadas dentro de una épica y estas corresponde a un breve requisito o funcionalidad que se desprende de la solución completa que podemos ir desarrollar en periodo de tiempo corto.

Una épica pueda contener varias historias de usuarios, ya que la funcionalidad a desarrollar podría desglosarse en múltiples requerimientos más pequeños y que sean independientes para que puedan desarrollarse sin que en algún momento una historia dependa de otra. Sin embargo, puede darse que existe dependencia en algún momento de que una historia sea terminada antes para poder iniciar otra y para ello se puede indicar que la historia de usuario es bloqueadora y se puede programar para que esta inicie antes.

Entre los campos que corresponder el ingresar tenemos:

- Título
 - Descripción
 - Fecha de inicio
 - Fecha de fin
 - Épica a la que está asociada
 - *Story points*
 - Criterios de aceptación (*Acceptance Criteria*)
 - Numero de *sprint*
- Subtarea

Las subtareas se encuentran agrupadas dentro de una historia y en ellas se detallan las tareas que se llevaran a cabo para completar la funcionalidad que

corresponde a la historia de usuario en la que se encuentra. Las subtareas se crean como pequeños grupos de trabajo que pueden ser medibles en un periodo de uno o dos días para poder medir el avance de la historia.

Aquí se debe agregar el detalle a bajo nivel de las actividades que realizaran y también pueden registrar el tiempo invertido en el desarrollo de la actividad de manera que se sepa con certeza cuánto tiempo lleva realizar algunas actividades y con ello poder estimar de mejor manera futuras tareas similares debido a la retroalimentación que se puede obtener de realizar este proceso.

Ente los campos de importancia tenemos:

- Título
- Descripción
- Fecha de inicio de tarea
- Fecha de fin de tarea
- Tiempo invertido en la tarea

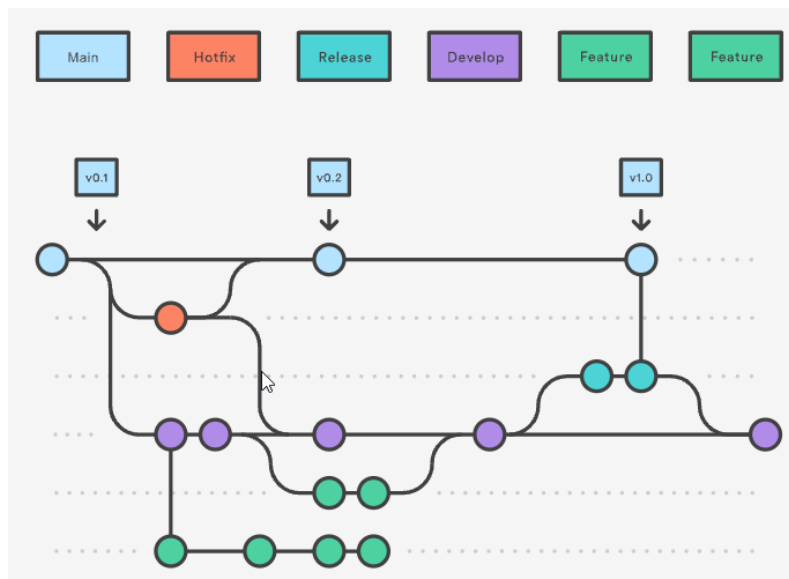
3.3. Manejo correcto de ramas

Las ramas o *Branch* como se conocen los manejadores de versionamiento permiten el poder tener el código de manera remota en un servidor en la nube y poder ir actualizando los cambios o bajando los últimos cambios disponibles dependiendo cual sea el caso.

Como parte de nuestro proceso de automatización es importante tener un manejo correcto del repositorio y de las ramas que irán surgiendo con forme vayan apareciendo nuevas funcionalidades a lo largo del desarrollo. Para ello también utilizando el modelo existente de manejo de ramas conocido como

Gitflow el cual brinda una serie de ramas principales conocidas como *Master* y *Develop* que serán las que contengan el código fuente funcional de las aplicaciones y ramas secundarias o para el desarrollo conocidas como *feature* las cuales partirán de *Develop* y se irán agregando conforme vayan surgiendo los desarrollos para posteriormente al terminar el desarrollo estas regresen a *Develop*.

Figura 4. **GitFlow**



Fuente: Atlassian (2021). *Flujo de trabajo de Gitflow*. Consultado el 05 de octubre de 2021.
Recuperado de <https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow>.

3.3.1. Nomenclatura correcta para ramas

La forma de nombrar las ramas es de importancia ya que esto ayuda a identificar de una mejor manera el código del desarrollo que se encuentra ahí. Para esta parte se recomienda seguir el siguiente esquema:

- Rama *master*

Esta rama es la que contiene el código de la aplicación que se encuentra en el ambiente productivo para esta rama se recomienda se mantenga su nombre en inglés y en minúscula como lo hace *Gitflow*.

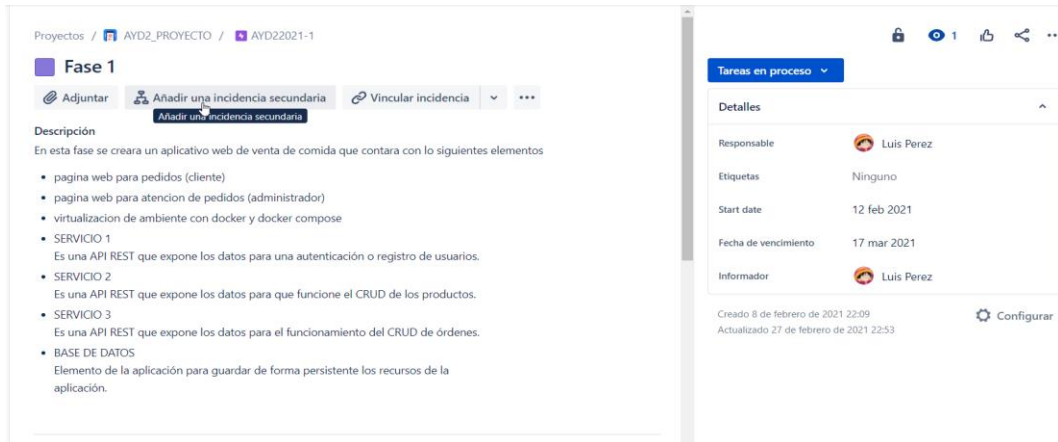
- Rama *develop*

Esta rama es la que contiene el código unificado de diversos desarrollos ya finalizado pero que no es productivo y que en él se pueden aun encontrar fallas. Para esta rama se recomienda que se mantenga su nombre en inglés y en minúsculas.

- Ramas *feature*

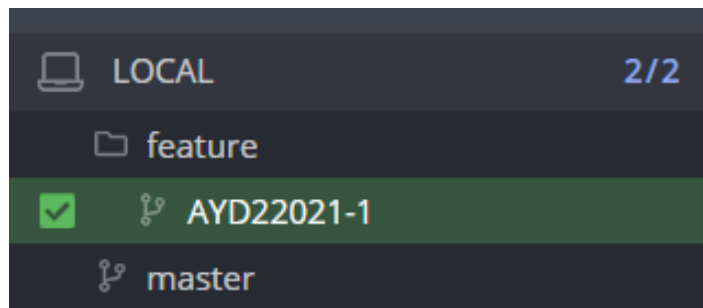
Estas ramas serán las que contendrá el código de las nuevas funcionalidades que se irán integrando a la aplicación para estas ramas se recomienda su nomenclatura sea la siguiente *feature/<correlativo_epica>* donde el *correlativo_epica* corresponde a código que se genera cuando se crea la épica en *Jira* para el desarrollo que a su vez es una combinación del nombre del proyecto y un valor numérico auto incremental que asigna *Jira* con forme se van creando épicas, historias de usuarios, entre otras.

Figura 5. Imagen de épica



Fuente: elaboración propia, realizado con *Greenshot*.

Figura 6. Creación de rama



Fuente: elaboración propia, realizado con *Greenshot*.

- Ramas de errores en pruebas internas

Quando se encuentra un error en la aplicación reportado por el equipo interno de *testing* o bien por los mismos desarrolladores y este se encuentre ya en la rama *develop* para solucionar dicha incidencia la nomenclatura apropiada

para este corresponde a `bugfix/<jira_issue_id>` donde el `jira_issue_id` es el correlativo que hace referencia a *bug* que se crea del lado de *Jira* este es como una historia de usuario, pero con la diferencia que el tipo cambia de historia de usuario a *bug*.

- Ramas de errores en producción

Un error en producción es un tema grave y que debe ser resuelto a la brevedad y para ello se tiene que tener presente que estas ramas no nacen de *develop* sino de quien posee el código de producción que es la rama *master* y por lo demás su nomenclatura es muy similar a la de los errores en ambientes de producción salvo el prefijo que se le antepone que corresponde a `hotfix/<correlativo_bug>` donde el `correlativo_bug` hace referencia el *bug* que sea crea en nuestro apartado en *Jira* y que es reportado del ambiente de producción del cliente.

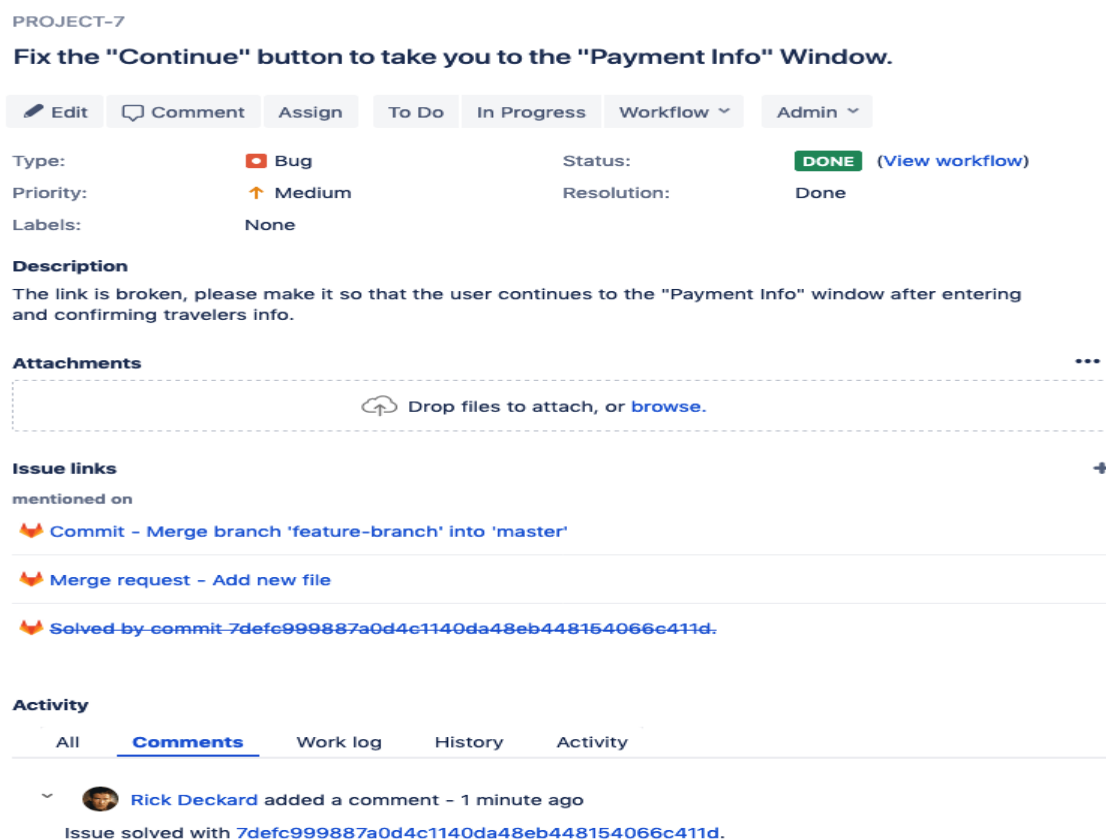
3.3.2. Nomenclatura correcta para *commit* en el proyecto

Los *commit* son parte del proceso de control de versiones en ellos se agrupan una serie de cambios realizados en el ambiente local en una rama local la cual al realizar el *commit* se sincroniza la versión local con la versión en la nube lo cual permite persistir los cambios de manera remota.

La razón del estandarizar la nomenclatura es que se pueda utilizar la información generada por el *commit* y los cambios asociados al para poder entrelazarlos con las tareas e historias en *Jira*, esto con la finalidad de si en algún momento necesitamos conocer el código desarrollado en una cierta tarea se pueda ir directamente a la tarea deseada y buscar el enlace que hace la referencia hacia el *commit*.

Para trabajar los *commit* con *jira* se necesita realizar una configuración tanto en la herramienta de *Jira* como en la de *Gitlab* esto para referenciarlas, por lo que como parte final a la hora de realizar la subida de los cambios a la nube en los *commit* debemos agregar un comentario con el comando `commit -m "<códigoTarea_espacio_texto_descriptivo_de_los_cambios>"` donde el `codigoTarea` corresponde al valor que se genera cuando se crea la tarea y posterior a ello agregamos una pequeña descripción de lo que hace el código.

Figura 7. **Commit enlazado con Jira**



The screenshot shows a Jira issue page for 'PROJECT-7'. The issue title is 'Fix the "Continue" button to take you to the "Payment Info" Window.' The issue is a 'Bug' with a 'Medium' priority and a status of 'DONE'. The description reads: 'The link is broken, please make it so that the user continues to the "Payment Info" window after entering and confirming travelers info.' Below the description is an 'Attachments' section with a dashed box and the text 'Drop files to attach, or browse.' The 'Issue links' section shows three links: 'Commit - Merge branch 'feature-branch' into 'master'', 'Merge request - Add new file', and 'Solved by commit-7defc999887a0d4c1140da48eb448154066c411d.'. The 'Activity' section shows a comment by Rick Deckard: 'Rick Deckard added a comment - 1 minute ago Issue solved with 7defc999887a0d4c1140da48eb448154066c411d.'

Fuente: Gilab (2021). *Jira integration issue management*. Consultado el 06 de octubre de 2021.

Recuperado de <https://docs.gitlab.com/ee/integration/jira/issues.html>.

3.3.3. Utilización de la función *rebase* en las ramas periódicamente

El uso de *git* en los proyectos de *software* es de vital importancia y más aún cuando el desarrollo que se está efectuando conlleva el involucramiento de un equipo de trabajo multidisciplinario y de gran envergadura, lo cual hace que al haber tantas personas involucradas dígase desarrolladores, *Dbas*, *DevOps engineer*, entre otro. Hacen que el mantener el orden en el controlador de versiones se vuelva una tarea compleja ya que al haber varios desarrolladores trabajando sobre una misma rama se puede dar el caso que dos o más desarrolladores realicen cambios sobre un mismo archivo lo cual a la hora de persistir los cambios generar lo que conocer como conflictos de código en el cual el gestor de versiones no sabrá que código debe persistir en la nube si el código A o el código proveniente B por lo cual solicitara se le indique si desea que sea uno u otro o ambos, ahora el problema en si no son los conflicto sino que esta tarea muchas veces es designada a un Release Manager que muchas veces puede que desconozca que realiza el código nuevo y el analizar dicho código para saber qué decisión tomar le puede llevar demasiado tiempo.

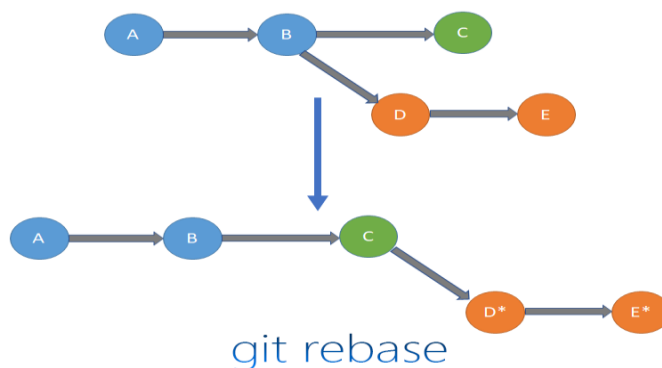
Debido a estos factores es que esta tarea debería de ser propia de quien realizo el código resolver estos conflictos y para ello se hará uso de una de las funcionalidades que ofrece *git* que es el *rebase*. El *rebase* permite tomar una serie de cambios ya confirmados en la rama actual y cambiar el punto de partida de este en base a la rama de la cual se originó lo cual permite que si se tiene una rama A la cual fue de donde partió nuestra rama B y con el pasar del tiempo a la rama A llegaron nuevos cambios que se podrían necesitar en la rama B los obtienen y se posicionan en el punto de partida de la rama ahora como el último cambio presente en la rama A. Con esto se consigue que el *commit* sea

trabajados de manera lineal uno tras otro sin que surjan *merges* innecesarios o ramificaciones.

Los pasos para realizar un *rebase* sobre una rama son los siguientes:

- Posicionarse sobre la rama que se desea realizar el *rebase*
- Sincronizar el repositorio local con el remoto utilizando *git fetch --prune*
- Crear una copia de la rama *git branch <nombre_actual>-<copia>*
- Iniciar el *rebase* con el comando *git rebase origin/develop* haciendo referencia a la rama sobre la cual queremos realizar la reorganización
 - Si existieran conflictos se resuelven y se hace el *commit*
 - Posterior se utiliza el comando *git rebase --continue* para continuar
- Si el *rebase* se procede con éxito se suben los cambios al repositorio remoto con el comando *git push --force-with-lease*

Figura 8. **Rebase en git**



Fuente: Cloudaffaire (2021). *Git Branching and Rebasing*. Consultado el 06 de octubre de 2021.

Recuperado de <https://cloudaffaire.com/git-branching-and-rebasing/>.

3.4. Definición de parámetros para aceptación de la calidad de código en *SonarQube*

Durante todo nuestro proceso de automatización y de implementación de *DevOps* se procura tener un nivel mínimo de aceptación en cada uno de los entregables y como parte de este proceso de control de calidad en el desarrollo es necesario el implementar una herramienta como lo es *SonarQube* la cual ayuda a analizar el código en base a reglas establecida para posteriormente poder detectar las posibles vulnerabilidades, *bug*, código cubierto, entre otros.

3.4.1. *Quality Profile*

Los QP o también conocidos como *Quality Profile* son parte fundamental de *SonarQube* pues ellos se tiene una serie de reglas agrupadas las cuales son las que servirán de indicador para alertar al momento de que alguna sea vulnerada se sepa y se pueda conocer que error o vulnerabilidad podría estar teniendo el código.

Los QP se encuentra asignado uno por cada lenguaje de programación que soporta la herramienta estos QP predefinido ya contienen una serie de reglas estándar para cada lenguaje y algunas reglas pueden ir variando de un lenguaje a otro, pero también se puede crear un QP personalizado y poder modificar acorde a los propios criterios de aceptación para cada uno los proyectos a manera de tener una plantilla definida para los lenguajes que se desea analizar.

Las ventajas que tenemos al utilizar las QP por defecto que tiene *SonarQube* para cada lenguaje es que nos permite ahorrar tiempo además que el nivel de las reglas que posee configurado ya brinda un muy buen nivel de seguridad, también en sus versiones más recientes estas QP la herramienta les

brinda un mantenimiento propio lo cual quiere decir que periódicamente actualizan las reglas, para ello se van disminuyendo la posibilidad de no poder tomar algunos nuevos tipos de errores en el análisis.

3.4.2. *Quality Gate*

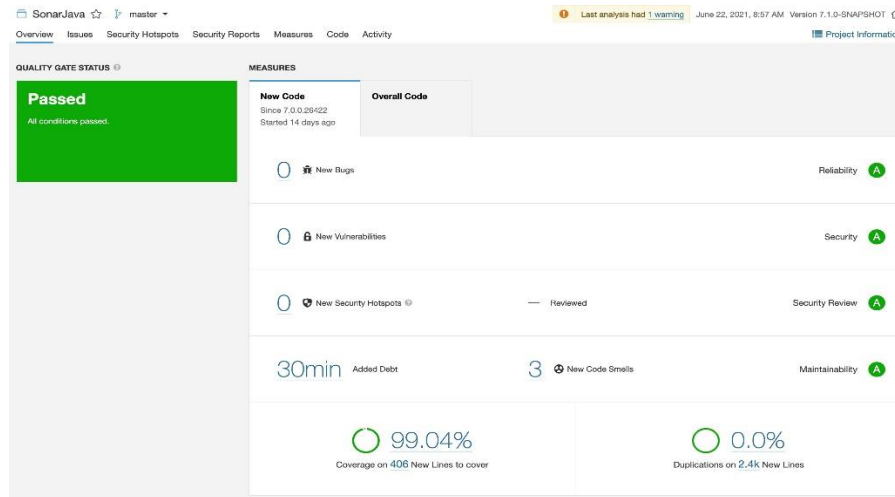
De la mano de los QP están los QG los cuales permiten crear reglas acordes a las necesidades en la cual se puede indicar alguna condición de calidad o de seguridad para posteriormente establecer un umbral de aceptación o de fallo.

En ellas se tendrán los parámetros para establecer si el código se encuentra bajo los parámetros que se han establecido como aceptables para la calidad de la aplicación y si es así pueda proceder a una siguiente fase del *DevOps* o por el contrario si algún parámetro está fuera de lo establecido por detener el flujo de nuestro *pipeline* ya que el código no cuenta con el nivel mínimo requerido para poder pasar a una fase de despliegue.

El QG por defecto que trae *SonarQube* cuenta con los siguientes valores ya establecidos:

- Cobertura de código: Es el valor porcentual del código que es cubierto por las pruebas unitarias siendo su valor por defecto el 80 %.
- Líneas duplicadas: Hace referencia a las líneas de código que se encuentran repetidas, haciendo una comparación de todo el código, su valor por defecto debe ser menos al 3 % del código total.
- Calificación global de fiabilidad: Esta no se representa mediante un porcentaje sino bajo una letra de calificación siendo A el valor aceptado en el cual se compara la cantidad de bug en base a su nivel de criticidad.

Figura 9. Tablero de **SonarQube**

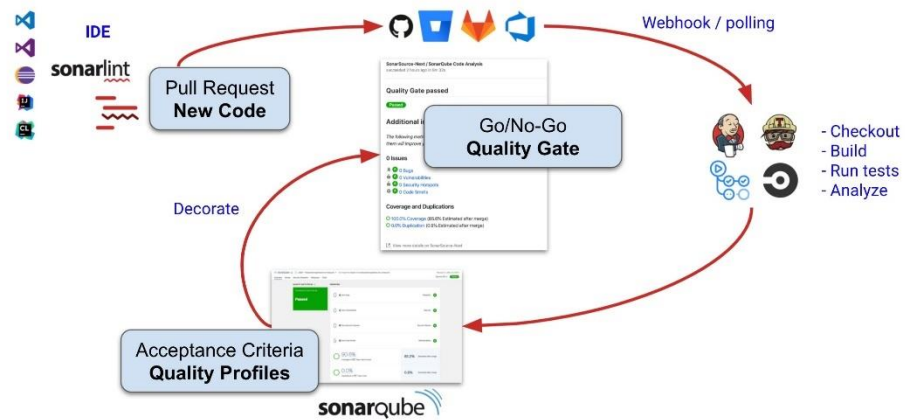


Fuente: SonarQube (2021). *The quality gate*. Consultado el 09 de octubre de 2021. Recuperado de https://blog.sonarsource.com/clean_coding-quality_profile_quality_gate_guidance

Es importante conocer cómo es que este flujo de trabajo inicia desde el *pull* del código al repositorio remoto desde nuestra computadora de trabajo para luego ejecutarse el *pipeline* que procederá a ejecutar el análisis el cual se podrá observar en el proyecto de *SonarQube*.

Figura 10. **Workflow SonaQube**

SonarQube Pull Request Analysis Workflow



Fuente: SonarQube (2021). *Quality Gate Application*. Consultado el 09 de octubre de 2021.

Recuperado de https://blog.sonarsource.com/clean_coding-quality_profile_quality_gate_guidance.

4. IMPLEMENTACION DE PROCESO *DEVOPS*

4.1. Planeación y diseño

Ahora corresponde la parte de la implementación del proceso de automatización de *DevOps* y se debe iniciar con las herramientas de la sección de planeación y diseño las cuales permiten empezar la gestión y diseño del proyecto por lo que se necesitan que sean de las primeras cosas a implementar.

4.1.1. Creación de espacio en *Jira*

Jira será la herramienta que permitirá gestionar muchas actividades del proyecto en ella se gestionara y se centralizará las historias de usuarios, épicas y lo correspondiente al manejo de entregables del proyecto en un periodo de tiempo determinado en la metodología que se decida adoptar para el proyecto.

Al momento de querer crear el espacio de *jira* se tienen varias opciones de las cuales se debió conocer un poco más en el capítulo de comparativas, sin embargo, si se está falto de experiencia en cuanto a la herramienta, pero se desea poder experimentar con ella se puede crear una cuenta gratuita en la cual se cuenta con una serie de beneficios como 10 usuarios por sitio, *agile reporting*, 2 GB de *storage*, entre otras.

Para crear el sitio de *jira* los pasos a seguir son los siguientes:

- Ingresar al sitio oficial de *Atlassian*
- Buscar la opción de *Jira Cloud Free*
- Se elige iniciar sesión con correo propio o con *Gmail*
- Al ingresar los datos de correo pedirá registra un nombre para nuestro sitio

Figura 11. Creación de sitio *Jira*

ATLASSIAN

Jira Software

Cloud Free

- ✓ Escala prácticas ágiles
- ✓ Consolida los flujos de trabajo
- ✓ Amplía la visibilidad
- ✓ Planificación, supervisión y publicación

Hola de nuevo, 2221899160101

Dirección de correo electrónico del trabajo *

2221899160101@ingenieria.usac.edu.gt

[Regístrate con una cuenta de Atlassian diferente](#)

Tu sitio *

.atlassian.net

Al hacer clic a continuación, aceptas las [Condiciones de servicio](#) de Atlassian Cloud y la [Política de privacidad](#).

Aceptar

NO SE NECESITA TARJETA DE CRÉDITO

Fuente: elaboración propia, realizado con *Greenshot*.

- Se valida que el nombre que se ingrese no exista previamente
- Se consulta el tipo de equipo que se maneja
- Se elige la profesión que más se acople a los miembros del equipo
- Se consultará de si son nuevos o expertos en el uso de la herramienta esto con la finalidad de ayudar con la configuración, esto desencadenará una serie de preguntas.

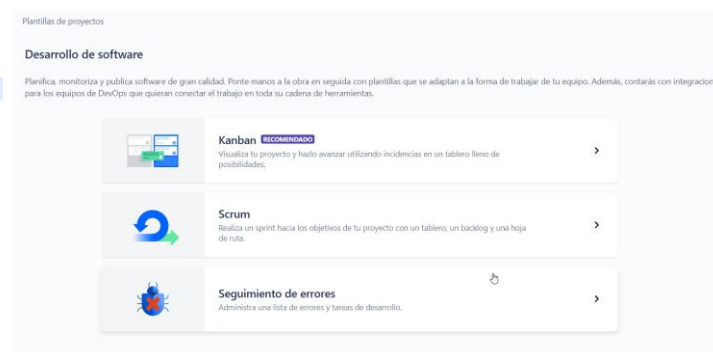
Figura 12. Ayudante de configuración



Fuente: elaboración propia, realizado con *Greenshot*.

- Se tendrá una serie de plantillas propias de *jira* que permitirán elegir bajo que formato se desea trabajar el proyecto siendo las opciones *Kanban*, *Scrum* o Seguimiento de errores.

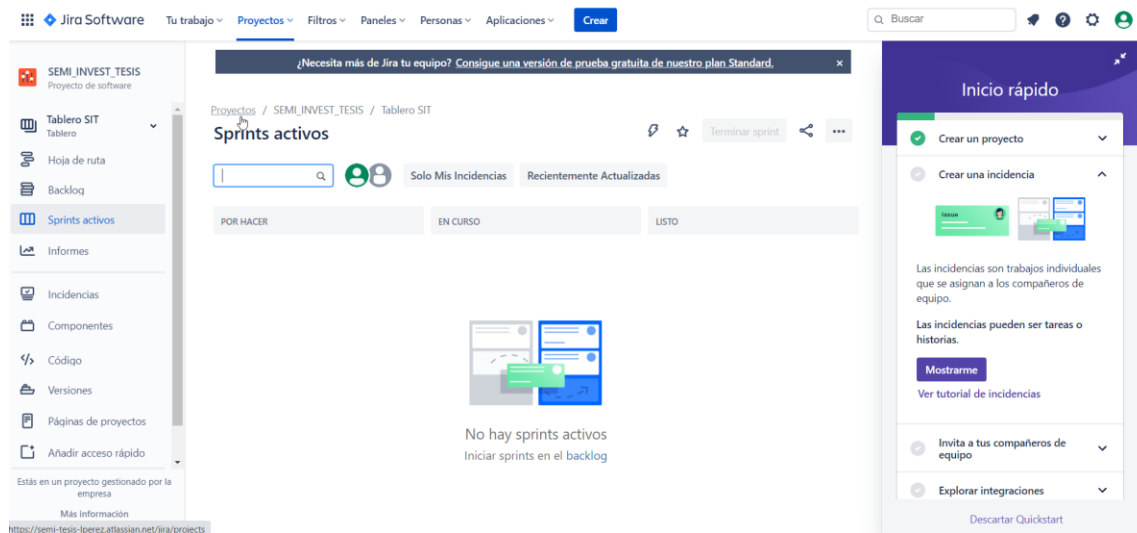
Figura 13. Plantilla inicial



Fuente: elaboración propia, realizado con *Greenshot*.

- Posterior a la selección de una plantilla se detallará la información de esta como los estados del *workflow*, tipo de incidencias, entre otros.
- Luego corresponde el elegir el tipo de proyecto entre los cuales brinda las opciones de Gestionados por la empresa o Gestionados por el equipo.
- Ahora se ingresa el nombre del primer proyecto.
- Se presiona el botón crear para poder tener el proyecto.

Figura 14. Proyecto en Jira



Fuente: elaboración propia, realizado con *Greenshot*.

Ahora se tiene el primer proyecto en la nube a través de *jira* y con el cual se podrá empezar a realizar la gestión de los desarrollos.

4.1.2. Creación de *confluence*

El *confluence* es la herramienta que se designado para el manejo y centralización de la información en el proceso de desarrollo y en lo que corresponde a la implementación de la automatización con *DevOps*.

Para realizar la configuración del espacio de *Confluence* se dirige siempre a la página de *Atlassian* e ingresan a la sección de *jira* en la cual nos solicitará las credenciales para cargar al *dashboard* de inicio de la página.

Ahora para configurar el espacio de *confluence* y poder ir documentando cada proceso se debe realizar los siguientes pasos:

- En la parte superior izquierda se tendrá un botón que al posicionarse sobre el dirá “cambiar a” en el cual se encontrará la opción de *confluence*
- Al seleccionar la opción desplegará una venta en la cual nos dará información general de que es *confluence* y se tendrá un botón que dirá “Probar ahora”

Figura 15. **Confluence**



Probar ahora

Conecta personas, ideas, planes y flujos de trabajo

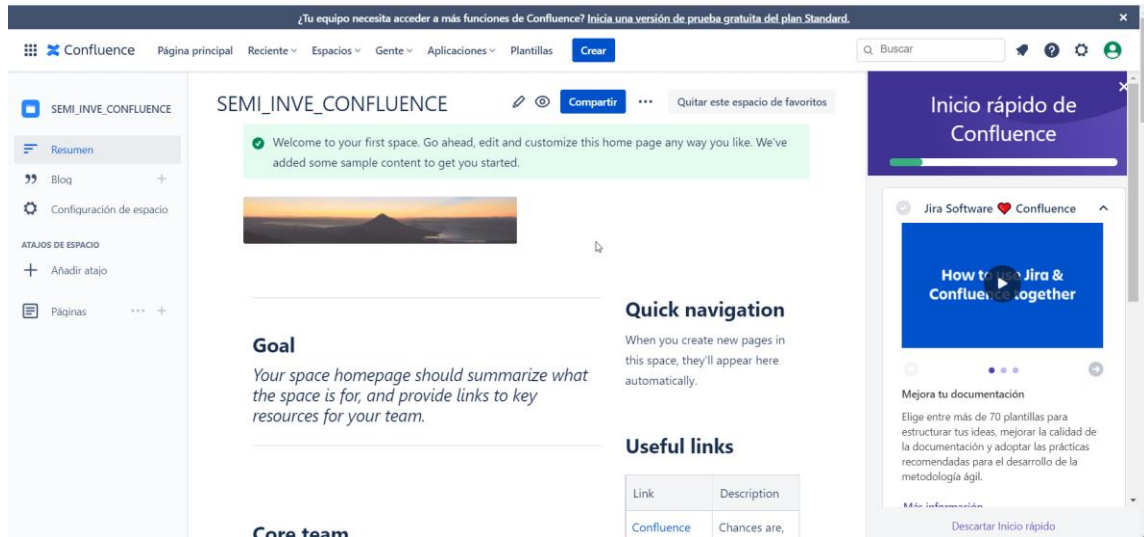
Confluence es un software de colaboración que facilita la creación de la documentación del proyecto y su mantenimiento en un único espacio. Basado en un marco abierto, Confluence ayuda a tu equipo a acabar con el aislamiento, a comunicarse de forma clara y a obtener una visibilidad total. Deja de buscar en el correo electrónico, en Google Drive y en las carpetas de Word, y empieza a encontrar lo que necesitas.



Fuente: elaboración propia, realizado con *Greenshot*.

- Al hacer clic en el botón nos mostrará un resumen de la información y los recursos disponibles que se tendrá entre los cuales están 10 usuarios que tendrán acceso, así como páginas ilimitadas.
- Solicitará de manera opcional el agregar personas al espacio o bien si se desea agregar a los miembros del proyecto en *Jira*.
- Ahora se empezará a gestionar nuestro espacio para el *confluence*.
- Se debe agregar cierta información si desea tener una configuración más acorde a las tareas o bien se puede realizar más adelante.
- Se debe agregar un nombre para el nuevo espacio.
- Finalmente tendrá el espacio listo para utilizar.

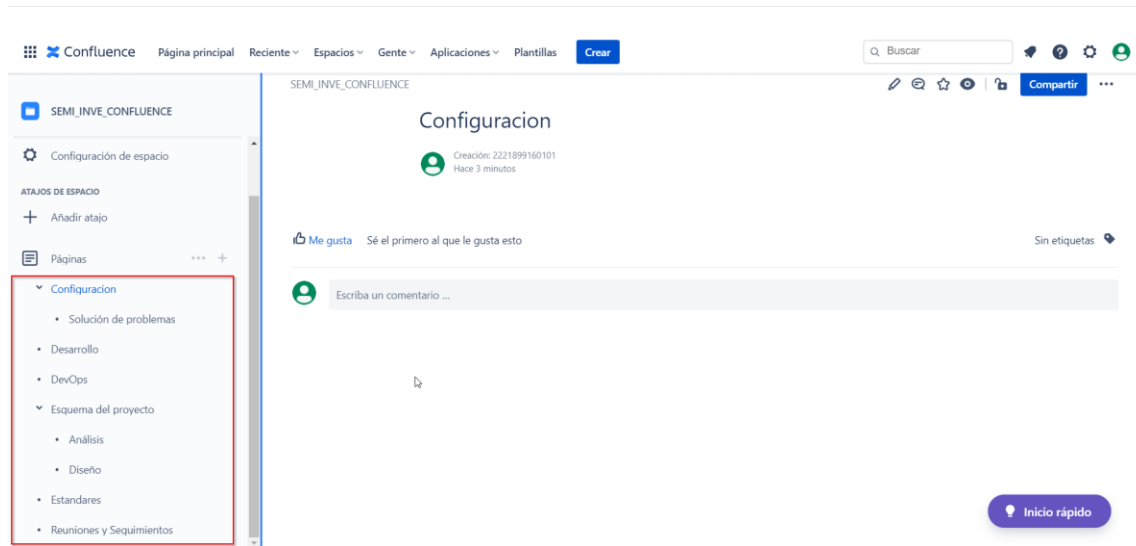
Figura 16. **Página inicial Confluence**



Fuente: elaboración propia, realizado con *Greenshot*.

- Ahora con lo definido en la sección 3.1 de la tesis en la cual se definió el manejo de la documentación, el espacio quedaría con la siguiente estructura.

Figura 17. **Plantilla de manejo de documentación**



Fuente: elaboración propia, realizado con *Greenshot*.

Ahora lo que corresponde es conforme se avance en el proyecto ir documentando en la sección correspondiente la información que se vayan generando para posteriormente sea consultada por los demás miembros asignados al espacio.

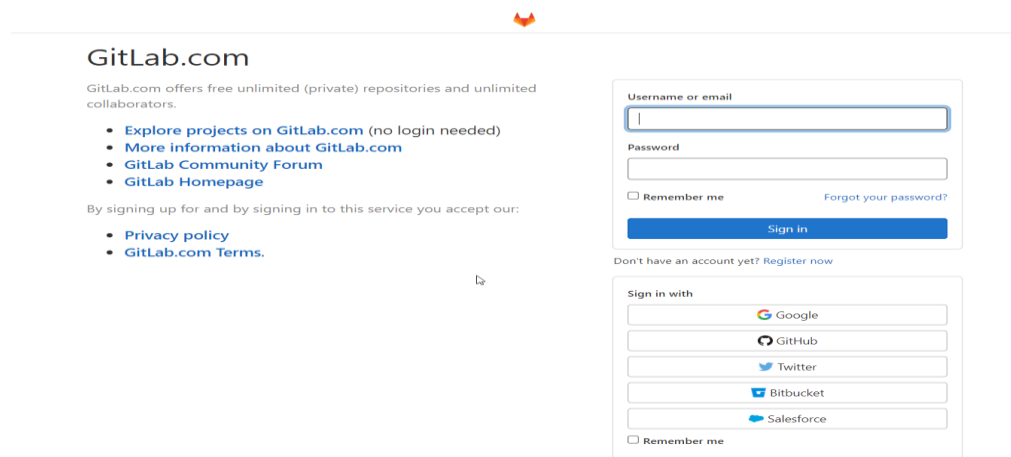
4.1.3. **Creación de repositorio en *GitLab***

Se ha tratado la comparativa de las herramientas de gestión y sobre las diversas herramientas existente basadas en *git*, se ha optado por *GitLab* debido a las ventajas que esta brinda y que se abordan en el capítulo 2, se conoce acerca de las herramientas que ya trae integrado para el desarrollo del proceso *DevOps* lo cual permite centralizar el código y el trabajo de automatización en una sola herramienta.

En *gitlab* también se trabaja *gitflow* para el manejo correcto de las ramas de los repositorios logrando con ello una mejor organización de donde se encuentra los desarrollos y en qué momento es oportuno el ir agregando estos nuevos desarrollos a las ramas *develop*, *master* y *release*.

Para crear una cuenta de *Gitlab* se debe ir a la página principal de *gitlab* en la cual se encontrará el botón login en la parte superior derecha. Esta opción redirigirá a una nueva página en la cual se tendrán las opciones de poder ingresar las credenciales si se posee alguna o bien crear una cuenta con los datos de una plataforma de terceros como *GitHub*, *Google*, *Twitter*, entre otras. Se debe seleccionar la que más convenga.

Figura 18. **Pantalla de inicio de sesión y creación de cuenta**



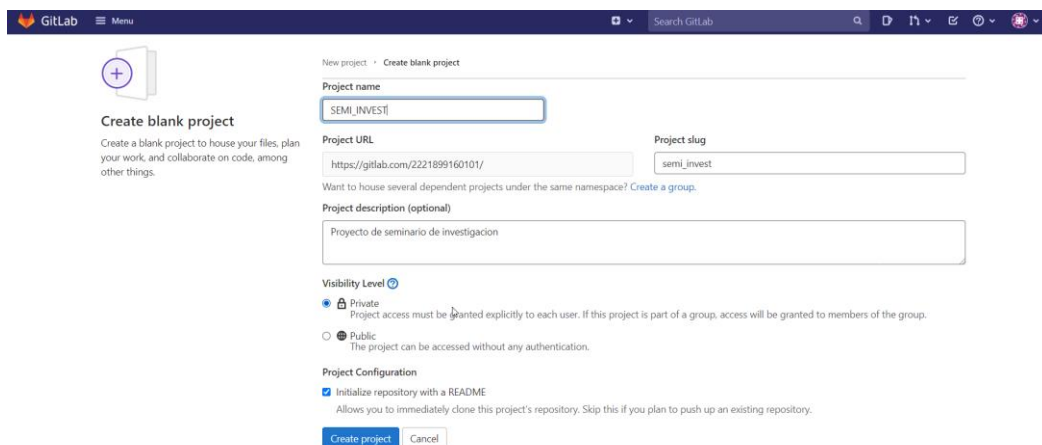
The screenshot shows the GitLab.com login and registration interface. At the top, it says "GitLab.com" and "GitLab.com offers free unlimited (private) repositories and unlimited collaborators." Below this, there are three links: "Explore projects on GitLab.com (no login needed)", "More information about GitLab.com", and "GitLab Community Forum". A note states "By signing up for and by signing in to this service you accept our:" followed by "Privacy policy" and "GitLab.com Terms". On the right side, there is a login form with fields for "Username or email" and "Password", a "Remember me" checkbox, and a "Forgot your password?" link. A blue "Sign in" button is at the bottom of the form. Below the login form, it says "Don't have an account yet? Register now". Underneath, there is a "Sign in with" section with buttons for Google, GitHub, Twitter, Bitbucket, and Salesforce, along with another "Remember me" checkbox.

Fuente: elaboración propia, realizado con *Greenshot*.

Creando la cuenta mostrará la página de inicio de *GitLab* en la cual se buscará la opción Menu y en ella se encontrará un apartado que dirá *Project* y este a su vez tendrá una opción que dirá *create new project*, al seleccionar esta opción enviara a una nueva ventana en la cual se tendrán las opciones para la creación de proyectos en las cuales se elige la de crear un proyecto en blanco.

Luego se encontrará en la pantalla de creación de proyecto en blanco en la cual se tendrá que agregar los valores indispensables para poder crear el primer proyecto los cuales corresponde a nombre del proyecto, se puede agregar una descripción la cual es opcional sin embargo se recomienda realizarlo para conocer un poco del proyecto sin entrar a profundidad, el nivel de visibilidad del proyecto el cual puede ser público o privado y con estos datos ya se puede crear el proyecto.

Figura 19. **Pantalla de creación de proyecto**



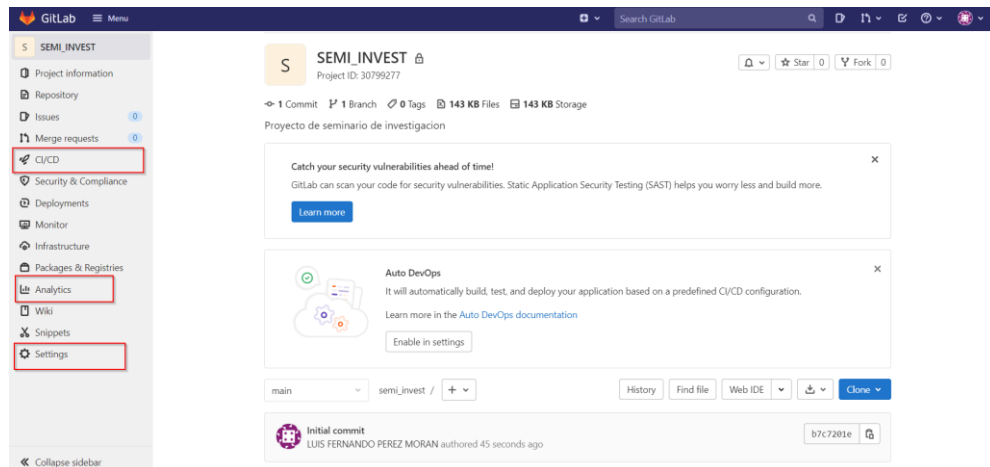
The screenshot shows the GitLab web interface for creating a new project. The page title is "New project · Create blank project". On the left, there is a "Create blank project" section with a plus icon and a brief description: "Create a blank project to house your files, plan your work, and collaborate on code, among other things." The main form contains the following fields and options:

- Project name:** A text input field containing "SEMI_INVESTI".
- Project URL:** A text input field containing "https://gitlab.com/2221899160101/".
- Project slug:** A text input field containing "semi_invest".
- Project description (optional):** A text input field containing "Proyecto de seminario de investigacion".
- Visibility Level:** Two radio button options: "Private" (selected) and "Public". The "Private" option has a sub-note: "Project access must be granted explicitly to each user. If this project is part of a group, access will be granted to members of the group." The "Public" option has a sub-note: "The project can be accessed without any authentication."
- Project Configuration:** A checked checkbox "Initialize repository with a README" with a sub-note: "Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository."
- At the bottom, there are two buttons: "Create project" (in blue) and "Cancel" (in grey).

Fuente: elaboración propia, realizado con *Greenshot*.

Como último punto se debe tener presente que en el proyecto de *GitLab* no solo ayudará en el manejo correcto del versionamiento del código sino también permitirá tener el manejo del proceso de automatización de *DevOps*. Aquí se podrá ir configurando las características necesarias para que el proyecto pueda empezar a generar los *pipelines* entre otros.

Figura 20. Pantalla de proyecto en *GitLab*



Fuente: elaboración propia, realizado con *Greenshot*.

4.1.4. Configuración de *runner* en *GitLab*

Ahora le toca el turno al *runner* de gitlab el *runner* como se sabe es el encargado de la ejecución de los *pipelines* del proyecto, es un recurso que se estará ejecutando fuera del ambiente propio de *GitLab* sin embargo no por ello deja de estar bajo el control de nosotros ya que al final el *runner* puede ser instalado en una máquina virtual del dominio en cualquier nube de preferencia.

Para la instalación del *gitlab runner* se necesita tener una máquina virtual con un sistema operativo *Linux* y que esta tenga acceso a internet y a su vez se pueda acceder a ella de manera remota mediante la *ip* publica y conexión *ssh*. Teniendo lo anterior ya configurado se procede a acceder a la terminal del sistema operativo, se ingresa en modo súper usuario y luego el siguiente comando “`curl -L --output /usr/local/bin/gitlab-runner https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-linux-amd64`” con lo cual procedemos a bajar los repositorios para la instalación.

Se agregan los permisos necesarios con el comando *chmod* “*chmod +x /usr/local/bin/gitlab-runner*” y se crea un usuario para el *runner* de *gitlab* “*useradd --comment 'GitLab Runner' --create-home gitlab-runner --shell /bin/bash*”, ahora se procede a instalar el *runner* utilizando el usuario creado “*gitlab-runner install -user=gitlab-runner --working-directory=/home/gitlab-runne*” y finalmente ponemos a correr el *runner* con el comando “*gitlab-runner start*”.

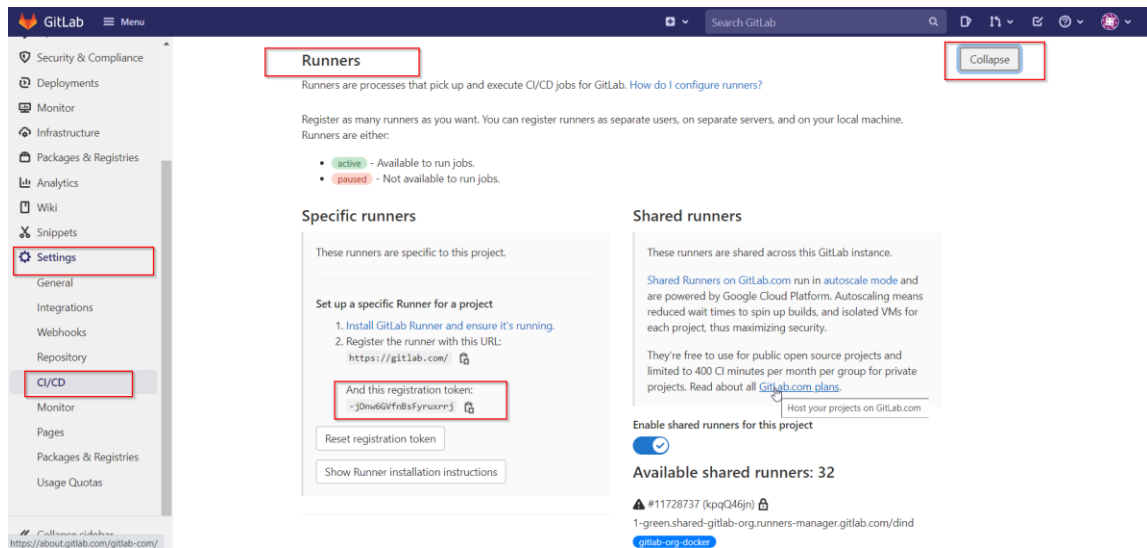
Ahora para vincular el *runner* se debe instalar en una máquina virtual, se debe ejecutar los siguientes pasos:

- Ejecutar el comando “*gitlab-runner start*”.
- Permitirá ingresar la url del nuestro *git* en nuestro caso por no ser en un servidor propio sino en la nube utilizamos <https://gitlab.com>.
- Solicitará ingresar un *token* el cual debemos obtener del proyecto de *gitlab*.

Para obtener el *token* del proyecto de *gitlab* y asociar el *runner* debemos seguir los siguientes pasos:

- Ir a la página de *gitlab*.
- Ingresar al proyecto y buscar la opción de *setting*.
- Buscar el apartado *CI/CD*.
- Luego en pantalla principal aparecerán varias opciones y se debe buscar la que diga *Runners*.
- Al presionar el botón *expand* el cual mostrara el *token*.

Figura 21. **Runners de gitlab**



Fuente: elaboración propia, realizado en *Greenshot*.

Ahora se tiene el *token* para continuar con la configuración que se dejó pendiente en el *runner* en la máquina virtual, para concluir con la configuración se debe realizar lo siguiente:

- Se ingresa el *token* para vincular el proyecto con el *runner*
- Se solicitará una descripción para el *runner* la ingresamos
- Por último, una parte esencial que dictara como manejaremos el *pipeline* la selección del tipo de *runner* entre los que tenemos *Shell*, *ssh*, *Docker*, entre otros.

La elección del tipo de *runner* dependerá de las necesidades del proyecto y de lo que se considere más conveniente ya que puede parecer la mejor opción el crear un *runner* de tipo *Shell* para ejecutar comandos *Shell* en nuestros

pipelines o bien uno *Docker* y poder aprovechar todas las ventajas de *Docker* en nuestra ejecución de *pipelines*.

4.1.5. Creación de tareas de desarrollo y *planning*

Ahora corresponde crear las tareas para el proyecto basándose en las categorías establecidas de Épica, historia de usuario, Subtarea, es importante tener presente que la cantidad de tareas a crear puede cambiar dependiendo el tipo de proyecto que se esté desarrollando y la metodología, ya que si se trabaja con una metodología como *Kanban* seguramente se tendrá que realizar la planificación de todo el proyecto y sus historias y como estas se deben desarrollar tanto en tiempo como en orden.

Si se aborda el proyecto con una metodología como *scrum* pues puede que se inicie con unas cuantas tareas a desarrollar que se podrían ir documentando al inicio de cada *sprint* en la reunión de *sprint planning* y también se sabrá que los tiempos tendrán un margen de 3 a 5 semanas.

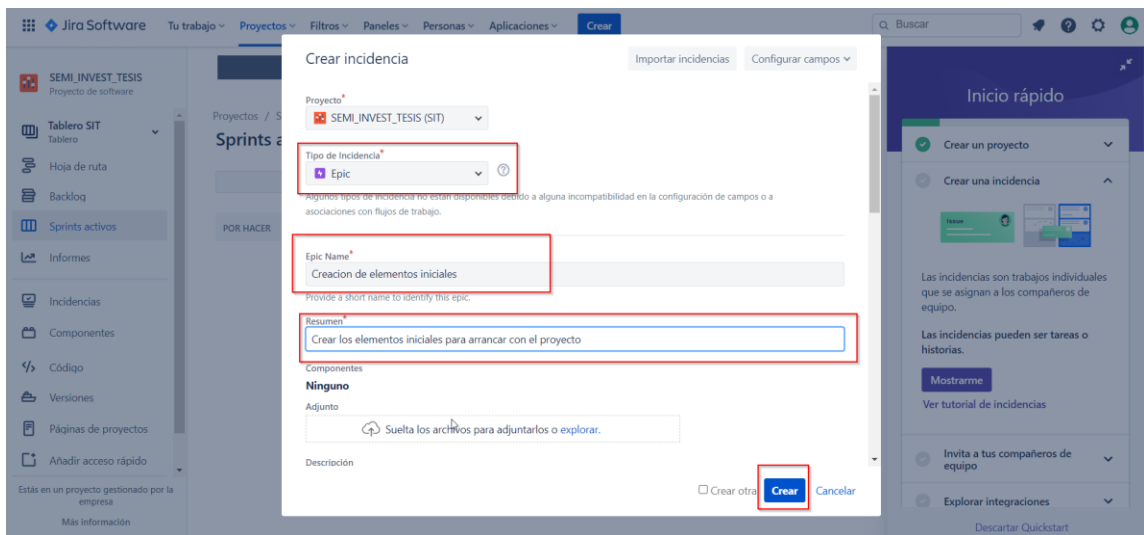
4.1.5.1. Creación de Épica

Para crear la primera épica y sus elementos asociados como lo son las historias de usuarios y las subtareas debemos realizar lo siguiente:

- Se ingresa a la página de *jira* y buscaremos el botón “Crear”.
- Este desplegará un modal en el cual se debe seleccionar el tipo de incidencia que para la primera será Épica.
- Requerirá agregarle un nombre a la épica.
- Un resumen el cual indicara la razón de ser de la épica.
- Se puede agregar opcionalmente una descripción.

- Se debe seleccionar la prioridad de la épica.
- Al llenar estos campos se presiona el botón “Crear”.

Figura 22. Creación de Épica



Fuente: elaboración propia, realizado con *Greenshot*.

4.1.5.2. Creación de Historia de usuario

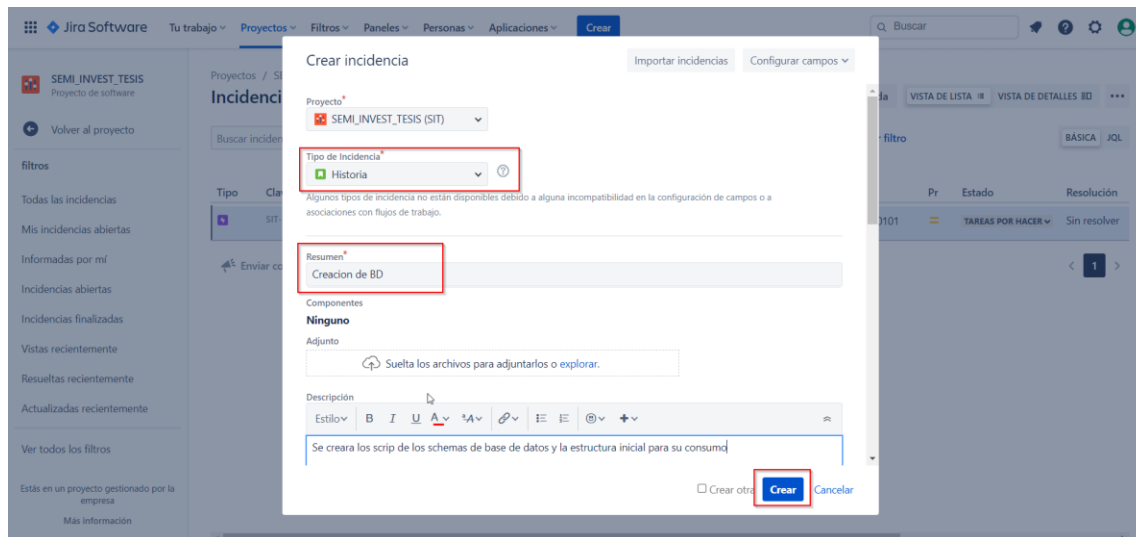
Se sabe que las épicas serán utilizadas para detallar una funcionalidad completa a desarrollar y las historias corresponde a una sub-funcionalidad que hace parte de la funcionalidad final.

Teniendo claro este concepto se procede a crear historias de usuarios que se pueda enlazar con la épica para lo cual debemos realizar lo siguiente:

- Se debe colocar en la página de inicio de jira y buscar el botón “Crear” como se realizó para la creación de la épica.

- Al desplegarse el modal la primera diferencia es que se elige que el tipo de incidencia sea “Historia” lo cual ara que aparezca una serie de nuevos campos.
- Se agrega el campo resumen que será el encabezado de la historia
- Se coloca el detalle correspondiente a la historia.
- Se asigna un responsable que será la persona a cargo de velar por el desarrollo de la sub-funcionalidad.
- Se le asigna una prioridad.
- En el campo *Epic Link* se asocia la historia a la épica correspondiente.
- También se asigna a un *sprint* si corresponde y si ya se tiene uno creado o en marcha.
- Al tener todos estos campos se presionar el botón “Crear”.

Figura 23. Creación de Historia de usuario



Fuente: elaboración propia, realizado con *Greenshot*.

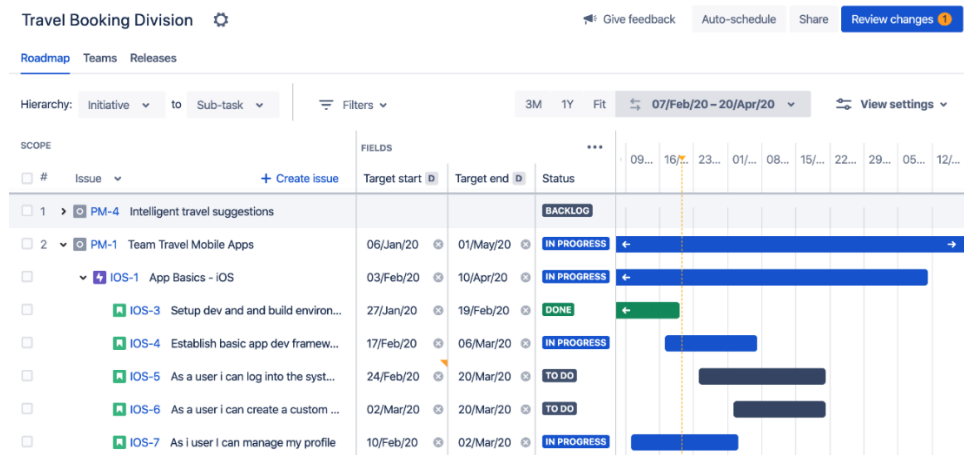
Realizados los pasos anteriores se deben crear tanto épicas e historias de usuarios como necesitemos para nuestro proyecto si utilizamos *kanban* o ir creándolo incrementalmente en cada *sprint*, todo dependerá de lo que mejor se adapte a las necesidades.

4.1.5.3. Utilización de *planning* de *Jira*

Una vez se tengan creadas todas las épicas con sus historias correspondientes y los tiempos asignados se podrá observar que *jira* creara una hoja de rutas o plan en el cual se podrá visualizar a manera de un *project* como se encuentra organizadas las actividades y el estado de cada una de las tareas.

Con ello se evita el tener que llevar en una herramienta de tercero una planificación de los tiempos, además se puede reflejar de manera más rápida si se tiene un atraso basándose en el tiempo transcurrido vs las tareas completadas.

Figura 24. *Planning* de *jira*



Fuente: Atlassian (2021). Creating plans. Consultado el 28 de octubre de 2021. Recuperado de <https://confluence.atlassian.com/jiraportfolioserver/creating-plans-952623582.html>.

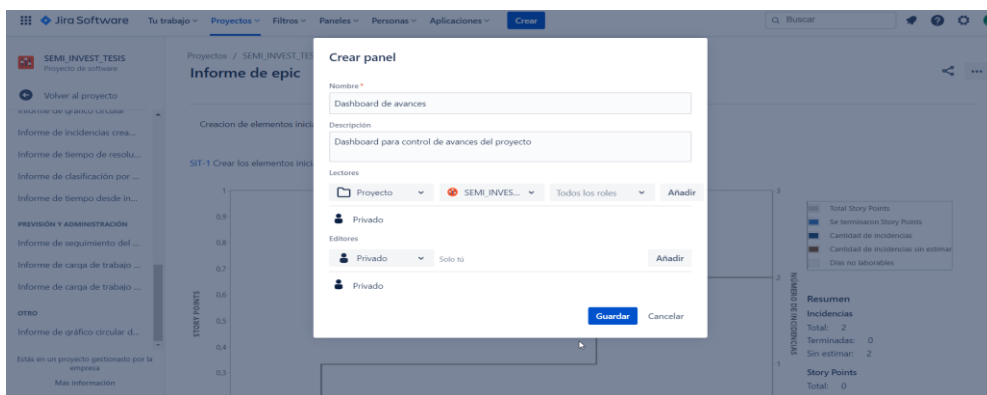
4.1.6. Creación de reportes para control de avances

La parte de gestión como vemos está compuesta diversas tareas que se deben ir realizando para ir formando el esquema del proyecto y de la automatización de este. Ahora corresponde la parte de la reportería la cual ayuda a ir visualizando información de importancia en el proyecto para medir tiempos, avances e ir conociendo el estado de las tareas en desarrollo.

Ahora se explica cómo crear un panel de control en el cual se pueda ir agregando las gráficas y reportería que se considere necesarias, para lo cual se debe realizar lo siguiente:

- Buscar la pestaña de “Paneles” y presionar la opción “Crear panel”.
- Se Ingresa un nombre y una descripción para el panel.
- Se selecciona las opciones para los lectores y quienes pueden modificar el panel.
- Ingresados los campos requeridos se presiona el botón “Guardar”.

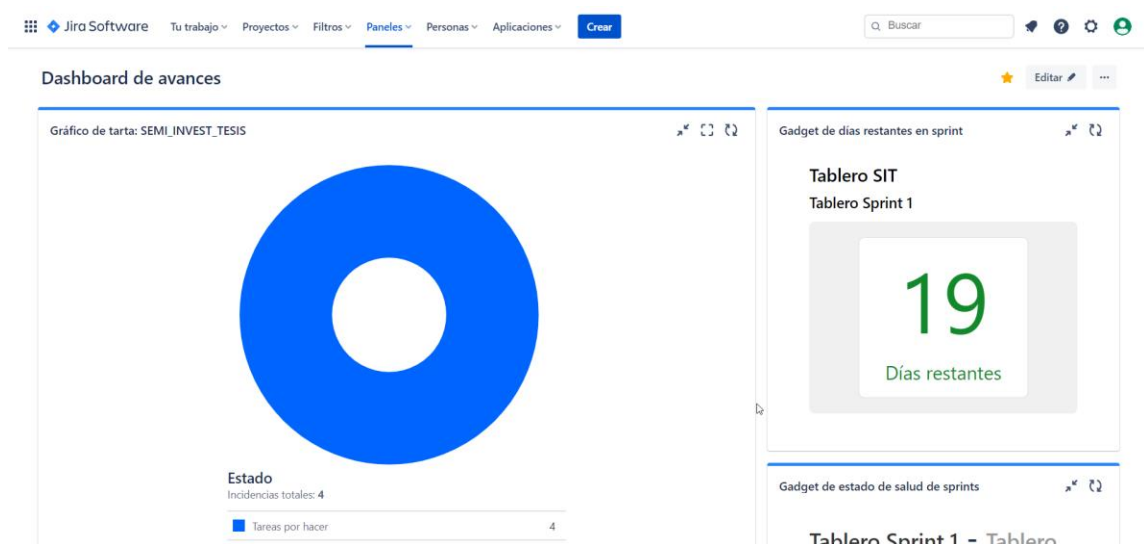
Figura 25. Creación de panel para reportes



Fuente: elaboración propia, realizado con *Greenshot*.

- Luego mostrara una pantalla con una serie de *gadget* con diferentes funcionalidades que podremos agregar a nuestro panel.
- Se selecciona y configura los *gadgets* con forme la información que cada uno nos solicite.
- Se presiona la opción de Guardar y ya se deberá tener el panel de avances y reportería.

Figura 26. Panel de reportería



Fuente: elaboración propia, realizado con *Greenshot*.

Ahora se podrá ir observando los días faltantes para el *sprint*, la cantidad de tareas y en qué estado se encuentran, la cantidad de tareas que tiene asignada cada miembro del equipo, entre otras.

4.1.7. Creación de *Kanban* de tareas

Primero se debe conocer dos elementos que ayudan al correcto uso de *Kanban* el primero es el *Backlog* el cual permite tener el listado de todas las historias de usuario que se desea desarrollar y del cual se podrán ir tomando elementos en nuestro *sprint planing* para cuando se inicie un nuevo *sprint*.

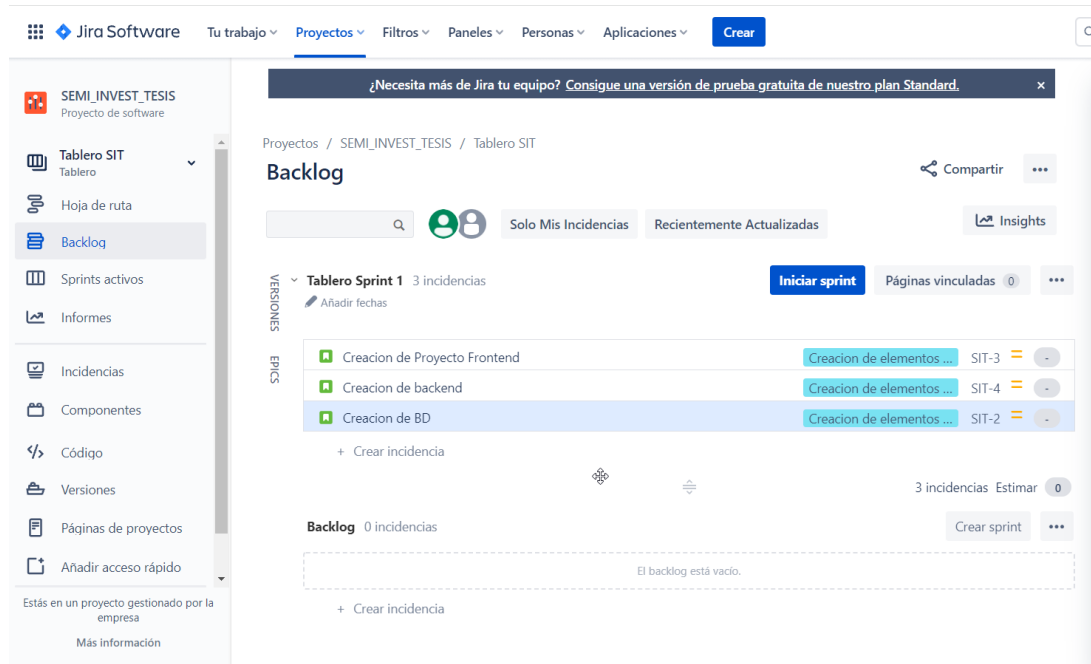
El segundo elemento corresponde al *sprint* como tal que para la herramienta que es *jira* es necesario crearlo e iniciar para poder agregar cada una de las historias de usuario que se desea sean desarrolladas en el *sprint* en curso. Estos elementos permitirán poder tener ya el tablero o *Kanban*.

El *Kanban* o también conocido como el tablero permitirá conocer de una manera más precisa que tareas e historias que se trabajaran y también ayudará a poder mover las tareas de un estado a otro de una manera muchas más fácil.

Los estados o *workflow* son las transiciones que sufre una épica, historia o tarea para poder llegar a un determinado estado o estatus. Para poder crear un *Kanban* se debe realizar lo siguiente:

- Se dirigen a la sección de *backlog*.
- Se encontrarán un botón que dirá Crear *Sprint*.
- Este nos habilitará una subsección en la cual se podrá tomar del *backlog* de historias las que necesitemos para iniciar el *sprint*.
- Una vez seleccionada todas las historias pertenecientes al *sprint*.

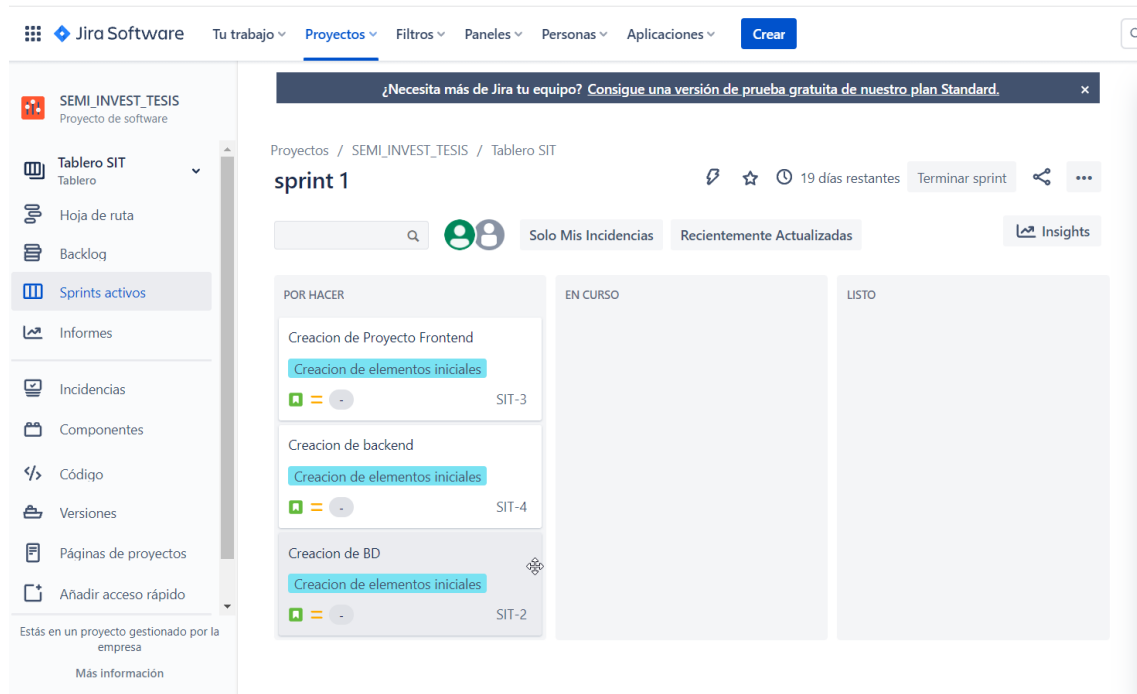
Figura 27. **Backlog jira**



Fuente: elaboración propia, realizado con *Greenshot*.

- Se presiona el botón *Iniciar sprint*
- Se debe nombrar el *sprint*
- Indicar la duración del *sprint*
- Se indica la fecha inicial
- Se agrega el *sprint goal*
- Con estos datos se puede iniciar el *sprint*

Figura 28. **Kanban sprint 1**



Fuente: elaboración propia, realizado con *Greenshot*.

Ahora se tendrá a disponible el tablero para poder ir moviendo de un estado a otro las historias y tareas. Con ello se logra una mejor visibilidad del avance de las tareas del equipo por cada *sprint*.

4.2. Creación

Ahora la configuración de los elementos con los cuales se creó cada uno de los componentes del proyecto. Básicamente ahora corresponde ver el entorno de desarrollo, la maqueta inicial del aplicativo *web* y el *branching model* para el control de versiones del código.

4.2.1. Configuración de *IDE* de desarrollo

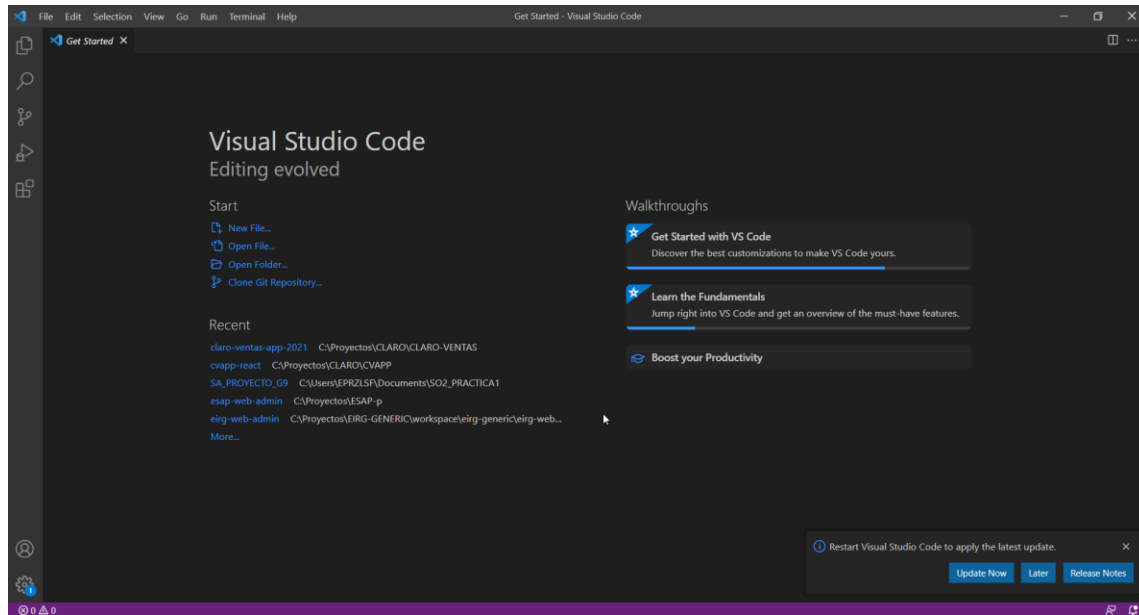
Se busca el desplegar una aplicación *web* mediante el proceso de automatización de *DevOps*, basándose en los temas tratados con anterioridad con respecto al desarrollo *web* se ha optado por desarrollar una aplicación basada en componentes con la librería de *react*.

En la actualidad uno de los mejores *IDE* para el desarrollo *web* es el creado por *Microsoft* llamado *Visual Code* el cual aparte de ser una herramienta de uso libre que brinda una gran cantidad de *plugging* los cuales pueden ayudar en las tareas de desarrollo, versionamiento de código, control de errores, entre otros.

Para configurar el *IDE* de *Visual Code* y dejarlo preparado para el desarrollo se debe seguir los siguientes pasos:

- Se dirigen a la página oficial de *Visual code* y descargamos la versión del sistema operativo que necesitamos.
- Descargado el *software* se inicia la instalación de la aplicación y se dejan las configuraciones predeterminadas.
- Se debe esperar un tiempo hasta que el *software* quede instalado en el equipo.
- Una vez finalizada la instalación se podrá acceder a al programa, al abrirlo tendrá una página de *Get Stated* en la cual tendrán atajos a proyecto abiertos recientemente.

Figura 29. **Visual Code**



Fuente: elaboración propia, realizado con *Greenshot*.

- Corresponde ahora la instalar las extensiones los cuales ayudarán en el desarrollo.
 - *Auto Close Tag*: Ayudara a cerrar las etiquetas *html* de manera automática.
 - *Material Icon Theme*: Ayudara a identificar de mejor manera los archivos y su tipo de manera más gráfica.
 - *Prettier – Code Formater*: Permitirá el formatear el código con la identificación correcta de manera automática al guardar cambios.
 - *Simple React Snippets*: Permite autocompletar los comandos de *react* ayudando a codificar de manera más rápida.
 - *SonarLint*: Permite enlazar con *SonarQube* para poder tener en cuenta las reglas configuradas para el proyecto.

Realizada estas las configuraciones se tendrá el *IDE* configurado para poder iniciar con las tareas de desarrollo, también es importante mencionar que *Visual Code* trae incorporado una serie de herramientas y funciones que pueden ser de utilidad como el que trae un complemento de *git* ya incluido el cual permite realizar *push*, *pull*, entre otros en el mismo *IDE* además cuenta con la opción de creación de terminales lo cual brinda la oportunidad de realizar comandos de consola desde la misma herramienta sin la necesidad de tener que abrir la terminal del sistema.

4.2.2. Creación de fuentes iniciales

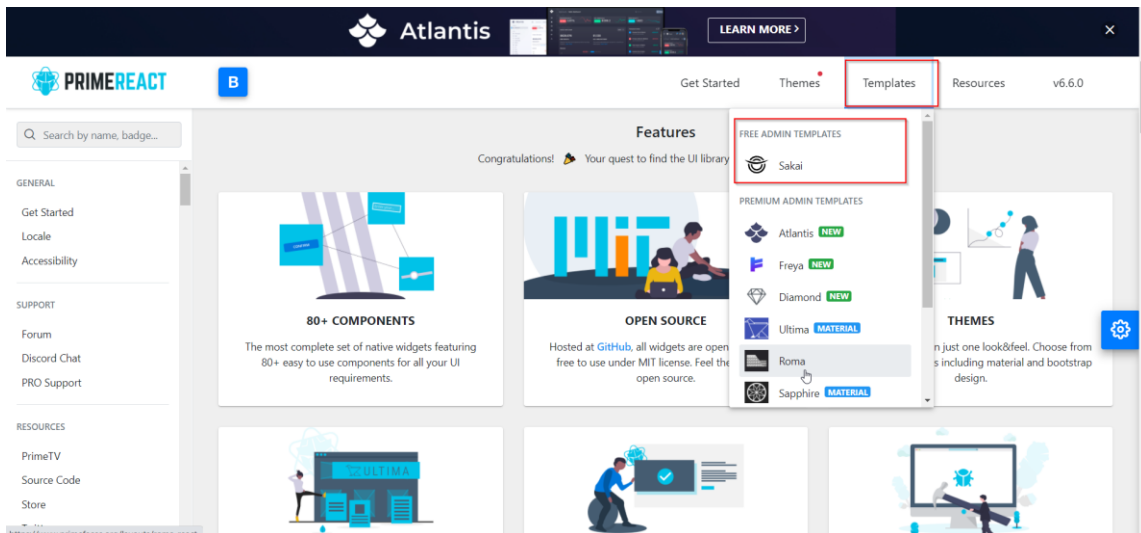
Configurado ya el *IDE* para poder realizar las actividades de desarrollo del proyecto. Ahora corresponde crear las fuentes iniciales para nuestro proyecto *web* para el cual se tienen las opciones de crear desde cero nuestro proyecto tanto archivos como la composición de las carpetas o bien utilizar alguna plantilla de pago o gratuita, esto dependerá de los recursos o las necesidades específicas.

Para poder crear el proyecto se optará por utilizar una plantilla gratis la cual se adquiere de la página oficial de *PrimeReact* la cual cuenta con sus propios componentes customizados para que se pueda utilizar por nosotros e ir creando también los componentes propios a base de estos.

Para crear nuestras fuentes iniciales realizaremos lo siguiente:

- Se dirige a la página oficial de *PrimeReact*.
- Se busca la sección de *template*.
- En la sección de *Free Admin Templates* tendremos los *template* disponibles de uso libre.
- Se selecciona la que más se acople a nuestras necesidades.

Figura 30. **Templates de PrimeReact**



Fuente: elaboración propia, realizado con *Greenshot*.

- Al tener descargadas las fuentes se recomienda realizar una copia de las fuentes originales.
- Tomar una de las copias de las fuentes y dejar solamente los archivos necesarios para iniciar el proyecto tales como:
 - El index.js o index.ts dependiendo el lenguaje
 - Los archivos de formato de css
 - El archivo App.js
- También se recomienda el manejar la siguiente estructura de archivos para poder estructurar el proyecto de una mejor manera.
 - *Component*: Archivos de componentes del proyecto.
 - *Pages*: Páginas de las que está compuesto el proyecto.
 - *Services*: Consumo de los servicios *rest* y *api* externas.

- *Theme*: Para el manejo de *css* y estilos.
- *DB*: para las conexiones a base de datos o consultas a *db* si fueran necesarias en el proyecto.
- *Utils*: Archivos con las utilidades que puedan ser necesarias en diversas partes del proyecto.

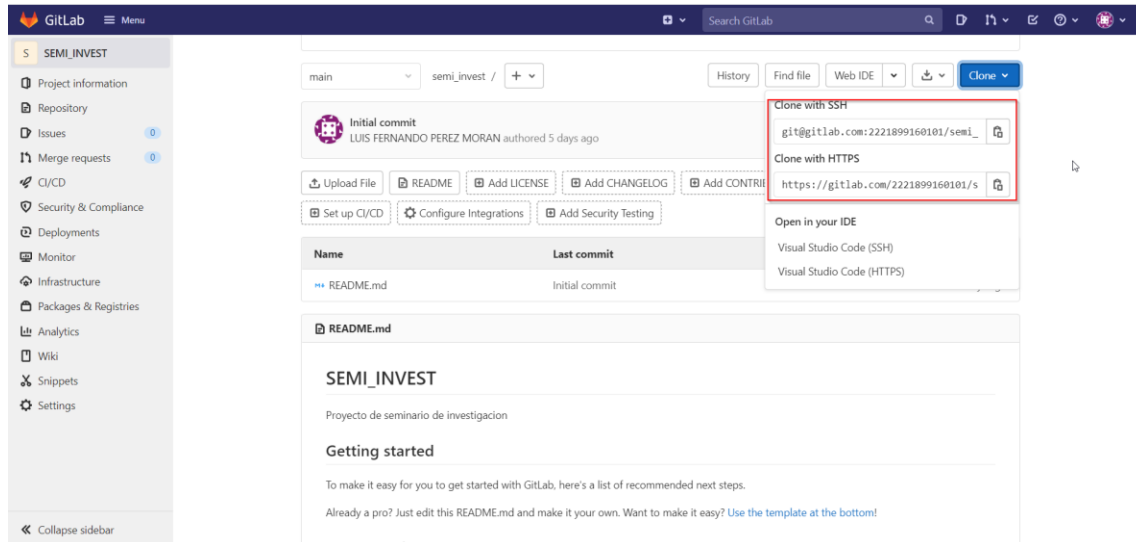
Realizadas cada una de estas configuraciones y estructuración de los archivos del proyecto se puede decir que ya se tiene la maqueta básica para poder iniciar a desarrollar. Es importante tener en cuenta que si el proyecto cuenta con más módulos de los cuales al inicio se irán desarrollando funcionalidades es necesario crear la estructura inicial de este, esto se puede suscitar si en el mismo proyecto se trabajara el *frontend* y *backend* de nuestro aplicativo y este *backend* fuera un *java spring* el cual se podría crear con ayuda de *eclipse* o alguna otra herramienta.

4.2.3. Creación de rama para desarrollo

Ahora que el *IDE* está configurado y las fuentes iniciales para poder desarrollar del proyecto se debe crear las ramas principales de las cuales partirá el desarrollo. En la sección 4.1.3 se creó el repositorio para el proyecto por lo cual debemos realizar los siguientes pasos para poder crear las ramas y subir las fuentes iniciales a la nube.

- Se dirigen a la página de *GitLab*.
- Buscan el proyecto e ingresan.
- Presionan el botón *Clone* el cual nos permitirá visualizar las opciones de clonación mediante *ssh* o *https*.
- Se selecciona la opción preferida.

Figura 31. Clonado de repositorio



Fuente: elaboración propia, realizado con *Greenshot*.

- Ahora se dirigen hacia la carpeta en la que se desea clonar el proyecto
- Se abre la consola de *git*.
- Debe teclear el comando *git clone <url_proyecto>*.
- Se tendrá ahora una copia local del proyecto.
- Ahora se copia a la carpeta clonada las fuentes iniciales.
- Regresa a la consola de *git* y teclea el comando *git add ** el cual permitirá agregar todos los cambios.
- Teclea *git commit -m "<comentario>"*.
- Como último paso ingresamos el comando *git push*.

Habiendo realizado cada uno de los pasos anteriores el resultado final será el haber clonado el repositorio en la rama *master* y haber subidos nuestras fuentes iniciales a esta rama. Se debe considerar que en versiones reciente de

git la rama *master* sufrió un cambio en su nombre y puede que aparezca con el nombre *main*.

Ahora para crear nuestra rama *develops* debemos seguir los siguientes pasos:

- Se abre la consola de *git* en el proyecto clonado.
- Se está ubicado en la rama *main* o *master* para lo cual teclearemos el comando *git brach <nombre_rama>*.
- Luego se posicionan sobre ella con el comando *git checkout <nombre_rama>*.
- Ahora se debe subir la rama al repositorio remoto para que esta puede ser accedida por el equipo con el comando *git push*.

Habiendo realizado esta serie de pasos se tendrá como resultado el haber creado la rama de *develop*, donde esta partió de *master* por lo cual tiene el contenido de las fuentes iniciales de *master*, así como que ya podrá ser accedida por los demás miembros del equipo ya que se encuentra en el repositorio remoto y a partir de esta rama podemos empezar a crear nuestras ramas de *feature* según lo establecido la sección 3.3.1.

4.3. Construcción

La construcción del proyecto iniciara con el *stage* de *build* del *pipeline*, el cual le corresponderá realizar la compilación previa para validar que el proyecto se pueda construir adecuadamente posteriormente al *merge* realizado de los desarrollos finalizados. Además, también permite tener un punto de control para saber si existe algún problema en la compilación previo al *deploy* de la aplicación.

4.3.1. Creación de *stage* de *build* en *pipeline*

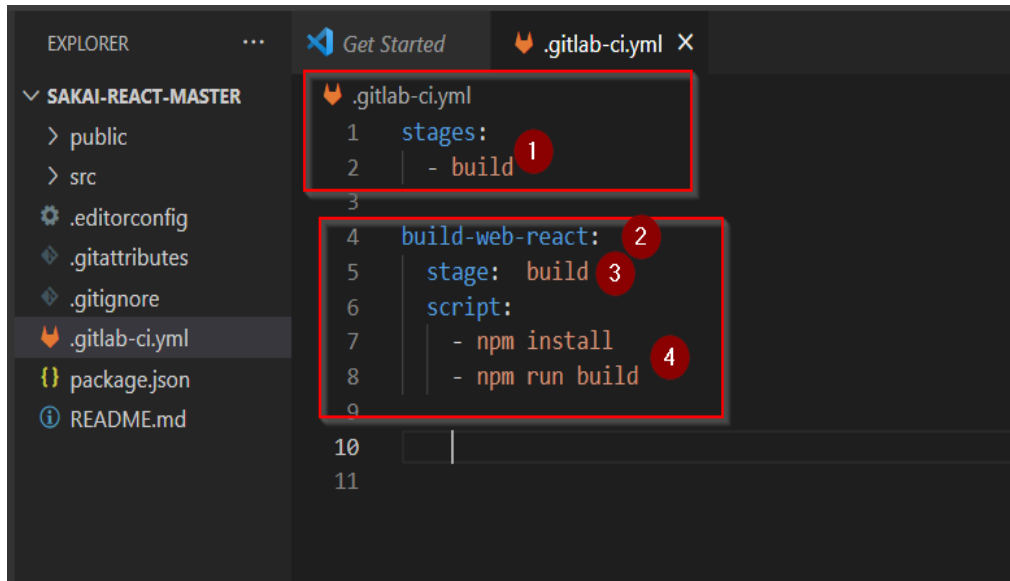
La construcción de nuestro *stage* de *build* se basa en que nuestro aplicativo corresponde a una aplicación *web* realizada con *react*, esto puede cambiar entre un proyecto y otro ya que puede haber diversas tecnologías incluidas en el desarrollo, pero la esencia de este *stage* se mantiene, siendo el punto principal el compilar nuestra aplicación.

Como primer punto se debe tener el archivo en el cual se codificarán las acciones necesarias para poder realizar nuestro *pipeline CI/CD*, para ello se necesita crear a la altura de la raíz del proyecto *web* donde se tiene el `package.js` el archivo `.gitlab-ci.yml` en el cual se maneja la lógica correspondiente a la implementación de *DevOps*. Este archivo se encuentra dividido en diversas secciones la cuales se abordará acorde se vayan necesitando y entro los cuales tenemos:

- *Stages*: Esta sección permite definir las diversas etapas por las que atravesara el código, estos se presentan como un listado de las palabras claves.
 - *Job*: Corresponde a una subdivisión de los *stage* y permite realizar una acción específica dentro del *stage*, estos son los encargados de ejecutar los *scripts* automatizados en el *runner*.
 - *Script*: Son el listado de las acciones a ejecutar de manera automatizada y en las cuales pueden contar con diversos comandos de consola.

El archivo `yml` al momento de quedar configurado para nuestro primer *stage* de compilación tendrá la siguiente apariencia.

Figura 32. Creación de *stage* de *build*



```
EXPLORER  ...  Get Started  .gitlab-ci.yml X
  SAKAI-REACT-MASTER
  > public
  > src
  .editorconfig
  .gitattributes
  .gitignore
  .gitlab-ci.yml
  package.json
  README.md

  1  stages:
  2    - build
  3
  4  build-web-react:
  5    stage: build
  6    script:
  7      - npm install
  8      - npm run build
  9
 10
 11
```

Fuente: elaboración propia, realizado con *Greenshot*.

Para poder explicar cómo se encuentra estructurado el archivo `.yml` se tiene que en la primera sección de *stages* se creó lo que corresponde al *stage* de *build* o compilación en español, en la siguiente sección ya hemos creado un *job* el cual se llama *build-web-react* el cual indicamos mediante la etiqueta *stage* que pertenece a *build* y finalmente con la etiqueta *script* ejecutaremos los comandos para instalar las dependencias y posterior compilar el aplicativo de *react*.

Algo importante a mencionar que para que funcione estos comandos es tener instalados tanto *node* como *npm* en el *runner* ya que las configuraciones que se realizaron en el *runner* fueron de consola, si en cambio se hubiese configurado como un *Docker* necesitaríamos agrega una etiqueta de imagen la cual nos indicaría que imagen de *Docker* se necesita para ejecutar cada *job* específico.

4.4. Calidad

Ahora entraremos en detalle a la construcción de nuestro *pipeline* en la sección de calidad del *software* en la cual se enfocan en dos puntos, en uno correspondiente a la ejecución de las pruebas unitarias y el otro a el análisis del código mediante *sonarqube*. Estos puntos permitirán validar que las funcionalidades desarrolladas cumplan con ciertos criterios de aceptación y que el código que se entregue no posea vulnerabilidades que permitan que este sea vulnerado en un futuro.

4.4.1. Creación de *stage* de prueba en *pipeline*

El *stage* de pruebas se enfoca en la ejecución de las pruebas unitarias las cuales pueden ser ejecutadas tanto para *backend* como para *frontend* esto dependerá de las exigencias del proyecto ya que puede que se opte por temas de recurso humano o factores monetarios solo realizar pruebas en el *frontend* o en el *backend*.

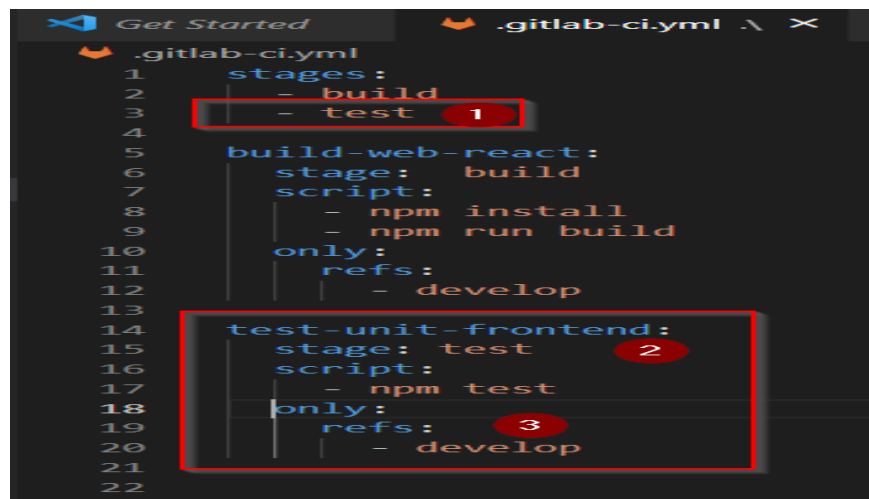
Para el proyecto se optó por el desarrollo de las pruebas del lado del *frontend* ya que se está realizando una automatización del *frontend* y este no posee un *backend* ya que se podría asumir que dicha implementación corresponde a otro proyecto por lo que los recursos se centran en la parte *web*. Por lo que es importante que para las pruebas se tendrá presente este punto y que al ser *web* se enfocaran en las pruebas unitarias utilizando la librería de *javascript* *chai.js* y el *framework* de *Mocha*.

Para las pruebas unitarias del *frontend* tenemos que tomar en cuenta que lo que se busca es validar que al realizar alguna acción aparezca algún elemento como un botón o que se renderize algún mensaje con un texto en específico en

una sección determinada de la *web* y no se busca el capturar algún valor en base a una acción que es enviada a un servicio para posteriormente ser tratados los datos y renderizados ya que eso correspondería a una prueba de integración y no una unitaria.

Para crear el *stage* de prueba se agrega una nueva etiqueta en la sección de *stages* la cual se llama *test* y posteriormente se agrega un *job* con nombre pruebas-unitarias-front en el cual se ejecutará los comandos necesarios para que se ejecuten las pruebas. También se agregará una nueva etiqueta llamada *only* que esta a su vez posee una llamada *refs* la cual permitirá restringir en qué momento queremos que se ejecute este *job*.

Figura 33. Creación de *stage* de prueba



```
1  stages:
2  | - build
3  | - test
4
5  build-web-react:
6  | stage: build
7  | script:
8  | | - npm install
9  | | - npm run build
10 |
11 | only:
12 | | refs:
13 | | - develop
14
15 test-unit-frontend:
16 | stage: test
17 | script:
18 | | - npm test
19 |
20 | only:
21 | | refs:
22 | | - develop
```

Fuente: elaboración propia, realizado con *Greenshot*.

Con los cambios que se realizaron sobre el archivo de configuraciones ahora se posee un *stage* más para la ejecución del *pipeline* el cual contendrá un *job* el cual ejecutara las pruebas unitarias y que ahora tiene la particularidad que

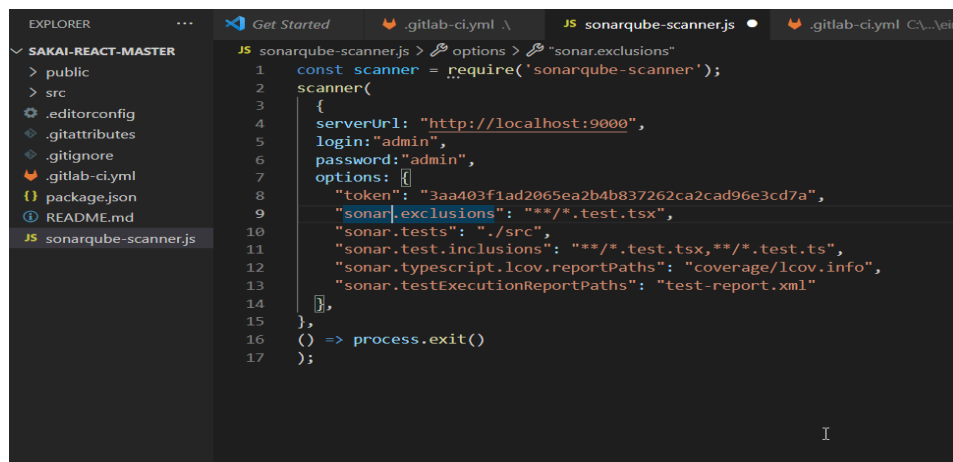
dichas pruebas solo se ejecutaran cuando se ejecute algún *merge* sobre la rama *develop*.

4.4.2. Creación de *stage* de *scan code* en *pipeline*

Ahora corresponde crear el *stage* para analizar del código y para lo cual se implementará la herramienta de *sonarqube*. Aquí se utilizan los estándares que se establecieron para el proyecto de la sección 3.4 en los cuales se tratan los temas de *Quality profile* y *Quality Gate*.

Para iniciar con el análisis se debe crear un archivo en el directorio raíz del proyecto, se llamará *sonarqube-scanner.js*, este archivo permitira el conectarse con el servidor de *sonarqube* y ejecutar el análisis del código, en él se podrá realizar configuraciones como usuario, contraseña, url de servidor de *sonarqube*, además también se podrá hacer exclusiones de archivos o analizar ciertas rutas del proyecto.

Figura 34. Archivo ejecución *sonar*

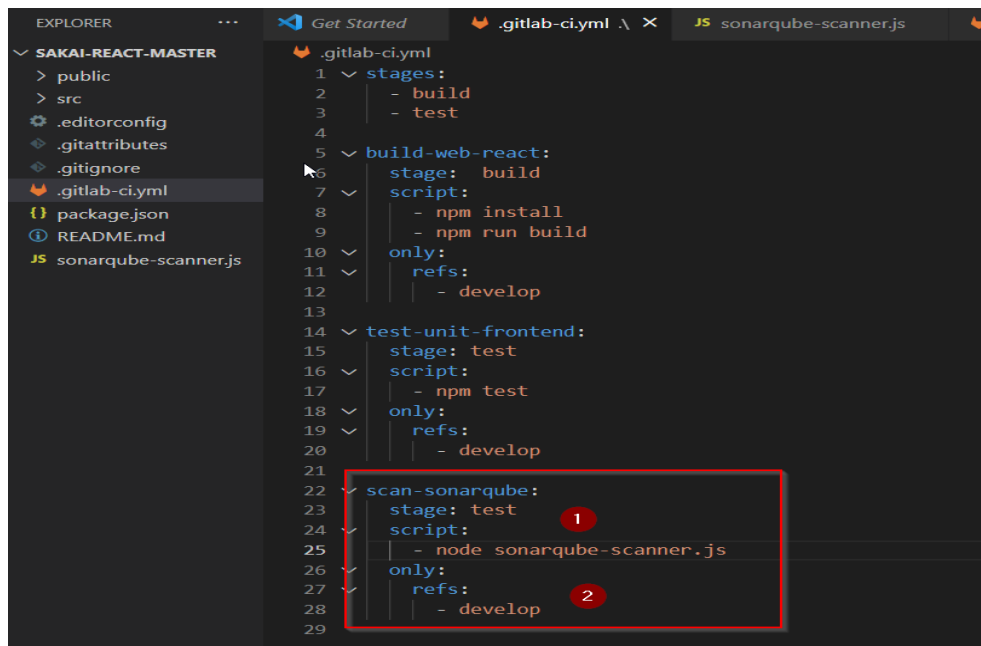


```
1 const scanner = require('sonarqube-scanner');
2 scanner(
3   {
4     serverUrl: "http://localhost:9000",
5     login: "admin",
6     password: "admin",
7     options: [
8       "token": "3aa403f1ad2065ea2b4b837262ca2cad96e3cd7a",
9       "sonar.exclusions": "**/*.test.tsx",
10      "sonar.tests": "./src",
11      "sonar.test.inclusions": "**/*.test.tsx,**/*.test.ts",
12      "sonar.typescript.lcov.reportPaths": "coverage/lcov.info",
13      "sonar.testExecutionReportPaths": "test-report.xml"
14    ],
15  },
16  () => process.exit()
17 );
```

Fuente: elaboración propia, realizado con *Greenshot*.

Creado el archivo de sonar se procede a la automatización de la ejecución del escaneo de código para lo cual se necesita crear un nuevo *job* el cual esté ligado al *stage* de *test*, al igual que los otros *jobs* se ara que este se ejecute únicamente cuando llegue a *develop*. Al final se tendrá un nuevo *job* llamado *scan-sonarqube* el cual generará el análisis del código y que posteriormente podrá visualizar en el proyecto de sonarqube.

Figura 35. Configuración escaneo de código *pipeline*



```
1  stages:
2    - build
3    - test
4
5  build-web-react:
6    stage: build
7    script:
8      - npm install
9      - npm run build
10
11  only:
12    refs:
13      - develop
14
15  test-unit-frontend:
16    stage: test
17    script:
18      - npm test
19
20  only:
21    refs:
22      - develop
23
24  scan-sonarqube:
25    stage: test
26    script:
27      - node sonarqube-scanner.js
28
29  only:
30    refs:
31      - develop
```

Fuente: elaboración propia, realizado con *Greenshot*.

4.5. Despliegue

En el despliegue de una aplicación se busca el poder tener los artefactos finales ya listos para poder ser ubicados en los servidores los cuales permitirán poder ejecutar la aplicación.

Para poder realizar el despliegue se debe saber qué tipo de artefacto se van a desplegar ya que se puede tener un *jar* el cual se necesitara desplegar ya sea en un servidor de aplicaciones como *web logic* o bien colocar en un servidor *Linux* o *window* y ejecutar un comando para que este se ejecute en el sistemas, también se podrá tener fuentes estáticas como las que genera un proyecto *web* como react o angular y colocarlas en un servidor *tomcat* o bien un *s3* de *Amazon* el cual permite deployar páginas *web*. Todo esto dependerá siempre de las particularidades de nuestro proyecto.

4.5.1. Creación de *stage* de *deploy*

Para poder deployar la aplicación *web* se debe tener presente que el artefacto que se genera en el *stage* de *build*, lo que nos generó fueron unas fuentes o archivos estáticos los cuales serán interpretados por el navegador *web* para que estos puedan ser visualizados posteriormente como una página *web*.

El destino de nuestro *deploy* puede ser desde un ambiente controlado de desarrollo hasta un ambiente de producción con el cliente final, se usara el *pipeline* para poder deployar el aplicativo en una máquina virtual de *Google Cloud Run*, también se podría hacer el despliegue en una maquina virtual propia de *Google cloud*, *aws* o bien utilizar algún servicio para *Amazon S3* que permite alojar un sitio *web* estático, así se puede dar muchos más ejemplos de diferentes *deploy* para una aplicación *web* pero se centraran en el servicio de *Google cloud run*.

Como primer paso se necesita crear nuestro servicio de *cloud run* para lo cual en la consola administrativa de *Google cloud* buscamos el servicio y de no tenerlo habilitado se procede a activarlo, posterior solicitara ingresar un nombre

para el servicio el cual le colocaremos despliegue-web-seminario a la cual se le asignara la imagen del contenedor que nos brinda por defecto el configurador.

Figura 36. Configuración de *Google Cloud run*

The screenshot shows the Google Cloud Platform console interface for configuring a Cloud Run service. The top navigation bar includes the Google Cloud Platform logo, the project name 'My First Project', and a search bar containing 'cloud run'. Below the navigation bar, there are two tabs: 'Cloud Run' and 'Crear servicio'. The main content area is titled '1 Configuración del servicio' and contains the following information:

- A descriptive paragraph: 'Cada servicio expone un extremo único y ajusta de forma automática la escala de la infraestructura subyacente para controlar las solicitudes entrantes. No se puede cambiar la región ni el nombre del servicio.'
- A radio button selection for 'Implementar una revisión desde una imagen de contenedor', which is currently selected.
- A text input field for 'URL de la imagen del contenedor *' with the value 'us-docker.pkg.dev/cloudrun/container/hello' and a 'SELECCIONAR' button.
- A section titled 'REALIZAR PRUEBAS CON UN CONTENEDOR DE MUESTRA' with a sub-note: 'Debe detectar las solicitudes HTTP en \$PORT y no depender del estado local.' and a link: '¿Cómo se compila un contenedor?'.
- A radio button selection for 'Implementar de forma continua revisiones nuevas desde un repositorio de código fuente', which is currently unselected.
- A text input field for 'Nombre del servicio *' with the value 'despliegue-web-seminario'.
- A dropdown menu for 'Región *' with the value 'us-central1 (Iowa)' and a link: '¿Cómo se selecciona la región?'.

Fuente: elaboración propia, realizado con *Greenshot*.

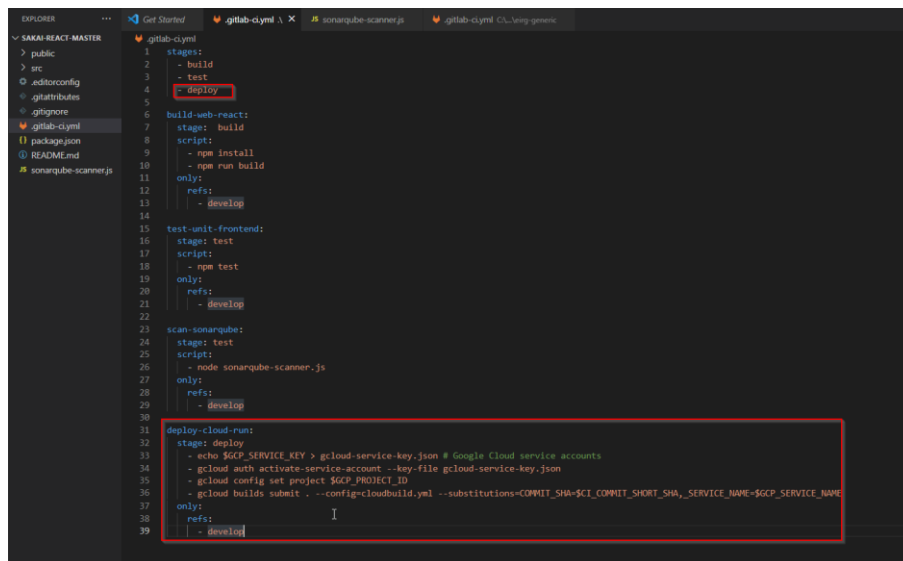
Realizadas las configuraciones anteriores corresponde activar la opción de permitir todo el tráfico lo cual permitirá poder recibir todo el tráfico entrante desde fuera de la red interna de la nube por lo cual se podría acceder al servicio desde la *web*, ya como último paso se procede a crear el servicio y esperar unos cuantos minutos en los que se crea el servicio.

Para el despliegue propio de la aplicación también necesitamos configurar en el *pipeline* la lógica que permitirá el poder llevar a cabo dicha tarea y para ello se configurará el *job* de *deploy* de la siguiente manera:

- Se creará un nuevo *stage* llamado *deploy*.
- Al *stage* de *deploy* se le creará un *job* el cual será para el *deploy* en *gcp*.
- Se utilizará la etiqueta de *image* para indicar que el *deploy* lo realizará mediante *Docker* con una imagen *cloud-sdk* la cual permitira realizar las acciones hacia nuestro servicio en la nube.
- Ya en la sección de *script* se agregarán los comandos de consola para automatizar el proceso.

Al finalizar dichas configuraciones se tendrá el archivo de *pipeline* listo para poder desplegar en nuestro servicio de *cloud run* nuestra aplicación *web*.

Figura 37. Configuración de *Google Cloud run*



```
1 stages:
2   - build
3   - test
4   - deploy
5
6 build-web-react:
7   stage: build
8   script:
9     - npm install
10    - npm run build
11
12 only:
13   - develop
14
15 test-unit-frontend:
16   stage: test
17   script:
18     - npm test
19
20 only:
21   - develop
22
23 scan-sonarqube:
24   stage: test
25   script:
26     - node sonarqube-scanner.js
27
28 only:
29   - develop
30
31 deploy-cloud-run:
32   stage: deploy
33   image: gcr.io/cloud-builders/docker
34   - gcloud auth activate-service-account --key-file gcloud-service-key.json
35   - gcloud config set project $GCP_PROJECT_ID
36   - gcloud builds submit . --config=cloudbuild.yml --substitutions=COMMIT_SHA=$CI_COMMIT_SHORT_SHA, SERVICE_NAME=$GCP_SERVICE_NAME
37
38 only:
39   - develop
```

Fuente: elaboración propia, realizado con *Greenshot*.

4.6. Verificación

Ya casi se ha completado un ciclo completo del proceso de automatización y como punto final se tiene la verificación la cual consiste en realizar pruebas que certifiquen que el producto final que se ha colocado en el ambiente productivo o de pruebas funcione de una manera correcta tanto la parte *web* como el *backend*.

Aquí se implementará los que son pruebas integrales para el *backend* creando pruebas que nos validen ciertos flujos que se consideren de importancia para la aplicación mediante la utilización de la herramienta de *SoapUI*, mientras que para la parte *web* de la aplicación haremos *test*, pero utilizando *RobotFramework* y *Selenium* los cuales permitirán validar que los componentes *web* se hayan renderizado adecuadamente y se encuentre donde corresponden.

4.6.1. Creación de *stage* de *automatic test*

Para las pruebas automatizadas podemos ejecutar una serie de pruebas tanto para el *frontend* y *backend*, para la parte *web* como lo que se ha tratado en otros apartados se ha decidido trabajar la automatización con *Selenium* y *RobotFramework*.

Para iniciar con la ejecución de las pruebas se tiene que haber previamente diseñado y creado las pruebas que se utilizaran para garantizar que nuestra aplicación *web* para ciertos flujos los elementos que se renderizan visualmente y los mensajes que se pueden visualizar mediante la interacción del usuario con la *web* son los correctos.

El primer paso como en otras sección es definir el nuevo *stage* el cual será el encargado de llevar a cabo estas tareas, el cual llevara por nombre test-

automation, luego de ello se procede a crear el primer *job* que corresponderá a la parte de la automatización *web* la cual llevara por nombre *front-end-test-automation* para luego en el *job* poder utilizar la etiqueta de *image* ya que se utilizara un contenedor de *Docker* para poder ejecutar en el las pruebas ya que al ser prueba correspondiente a sobre una página *web* se necesita de la utilización de un navegador *web* para que estas se ejecuten correctamente.

Para poder ejecutar las pruebas también se necesita utilizar la etiqueta de *before_script* la cual permite ejecutar comandos previos a lanzar los comandos principales del *job*, aquí se buscará realizar la instalación de la librería de *robot* con *selenium* mediante comandos de *pytom* ya que *selenium* es dependiente de *pytom* para poder ejecutarse. Realizadas las instrucciones anteriores ya se puede ejecutar los comandos principales para la ejecución de las pruebas los cuales se ejecutarán utilizando la etiqueta *script*.

Figura 38. Creación de *job* de *test automation*

```
32 deploy-c1oud-run:
33   stage: deploy
34   - echo $GCP_SERVICE_KEY > gcloud-service-key.json # Google Cloud service accounts
35   - gcloud auth activate-service-account --key-file gcloud-service-key.json
36   - gcloud config set project $GCP_PROJECT_ID
37   - gcloud builds submit . --config=cloudbuild.yml --substitutions=COMMIT_SHA=$CI_COMMIT_SHA
38 only:
39   refs:
40     - develop
41
42 front-end-tests-dev:
43   stage: test-automation 1
44   image: ppodgorsek/robot-framework 2
45   before_script:
46     - python3 -m venv env
47     - source ./env/bin/activate 3
48     - python -m pip install robotframework-selenium2library
49     - python -m pip install --upgrade pip
50   script:
51     - cd seminario-frontend-automated-tests
52     - robot -d results -i TestPlan1 4
53   only:
54     refs:
55       - develop 5
56   when: manual
```

Fuente: elaboración propia, realizado con *Greenshot*.

Ahora mediante la ejecución del proceso de automatización se puede garantizar que al tener desplegado el aplicativo *web* en el servidor o servicio *web* este tendrá el comportamiento esperado debido y que la interfaz del usuario contará con los elementos visuales para interactuar correctamente con las pantallas.

4.6.2. Creación de *stage performance*

Como parte final de del proceso se realizará la ejecución de las pruebas de *performance* las cuales ayudan a poner carga o estrés a la aplicación y sus servicios *rest* con el fin de conocer los tiempos de respuesta y ver como esta se comporta al tener una carga de peticiones elevadas que deba de atender.

Esto permite conocer a profundidad tiempos de respuestas, cantidad de transacciones procesadas, erróneas, entre otras. Se pueden realizar una serie de pruebas que ayudarán a garantizar la transaccionabilidad de la aplicación si esta es una aplicación altamente transaccional.

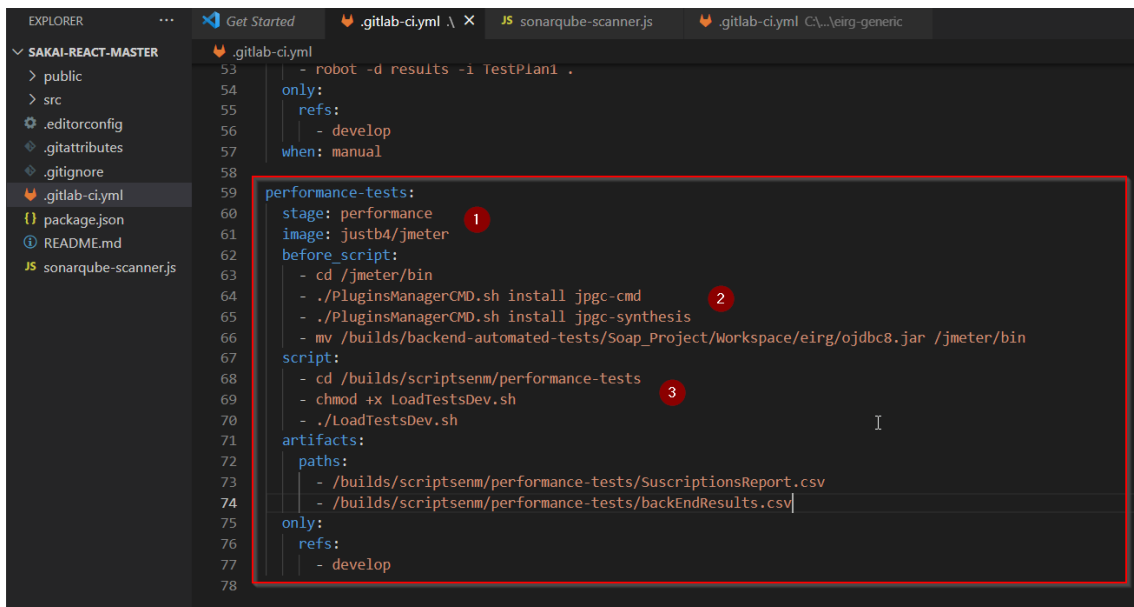
Para la creación de la automatización de *performance* se necesita crear el *stage* de *performance* y luego agregar un *job* el cual corresponderá a la ejecución de las pruebas llamado *performance-test*, además se utilizará la etiqueta *image* que permite asociar una imagen de *Docker* que en este caso correspondería a una de *jmeter* para posteriormente con la etiqueta de *before_script* poder ejecutar comandos de la consola para mover de ubicación el *jar* de conexión de base de datos.

Ya realizadas las configuraciones previas a la ejecución de las pruebas de *performance* se configura la sección de la etiqueta *script* en la cual incluiremos

los comandos para ubicar donde se tendrá el archivo *sh* y darle los permisos necesarios para que este se ejecute adecuadamente.

Finalizada estas configuraciones en el archivo de gitlab-ci para la parte de *performance* tendremos el siguiente resultado.

Figura 39. Creación de *job* de *performance*



```
53 - robot -d results -i TestPlan1 .
54
55 only:
56   refs:
57     - develop
58   when: manual
59
60 performance-tests:
61   stage: performance
62   image: justb4/jmeter
63   before_script:
64     - cd /jmeter/bin
65     - ./PluginsManagerCMD.sh install jpgc-cmd
66     - ./PluginsManagerCMD.sh install jpgc-synthesis
67     - mv /builds/backend-automated-tests/Soap_Project/Workspace/eirg/ojdbc8.jar /jmeter/bin
68   script:
69     - cd /builds/scriptsenm/performance-tests
70     - chmod +x LoadTestsDev.sh
71     - ./LoadTestsDev.sh
72   artifacts:
73     paths:
74       - /builds/scriptsenm/performance-tests/SuscriptionsReport.csv
75       - /builds/scriptsenm/performance-tests/backEndResults.csv
76   only:
77     refs:
78       - develop
```

Fuente: elaboración propia, realizado con *Greenshot*.

Finalmente, el ciclo completo de ejecución de automatización del cual se ejecuta el análisis de *performance* para conocer los tiempos de respuesta de los servicios rest, a esto le sumamos el hecho que también se agrega estrés a la base de datos y servidores por lo que también se puede tener pruebas de estrés con volúmenes altos de servicios *rest* siendo consumidos para conocer el máximo de peticiones que soporta la aplicación trabajar.

5. PLANTILLAS

5.1. Plantilla de estructura de nuevo proyecto en *jira*

Una plantilla permite definir una estructura la cual ayudara a definir parámetros necesarios para la creación de un nuevo proyecto aquí se definen los parámetros mínimos y obligatorios para tener presente siempre que se necesite iniciar con un nuevo proyecto de cualquier tipo basándonos en los estándares definidos.

Tabla IX. **Plantilla de propiedades *Jira***

Atributo	Descripción	Nomenclatura	Cardinalidad
Nombre Proyecto	Sera el identificador del proyecto el no interno es opcional por si dentro la organización se lleva un código de proyecto	<siglas proyecto>-<no interno> Ejemplo TESIS_0079	1
Abreviatura	Abreviatura con la que aparecerán las épicas e historias creadas	<siglas que describan el proyecto en mayúscula> Ejemplo TS0079	1
Entrega	Se utiliza la nomenclatura estándar para numera las entregas las cuales pueden tener agrupadas épicas o historias	<numero de version> Ejemplo 0.1 = primera versión 1.0 = versión producción	1..*

Continuación de la tabla XVII.

Sprint	Permite agrupar funcionalidades a entregar	<sprint_noSprint> Ejemplo sprint_01	1..*
Épica	Permite agrupar una serie de funcionalidades o desarrollos	<abreviatura>_<Titulo_Funcionalidad _general> Ejemplo TS0079_CREACION_INDICE TS0079_MARCO_TEORICO	1..*
Historia de usuario	Permite tomar la funcionalidad principal y segmentarla en una funcionalidad más granular dentro de una épica	<abreviatura>_<Titulo_Funcionalidad _general> Ejemplo TS0079_TEMA1 TS0079_TEMA2	1..*
Sub tarea	Permite desglosar una historia en pequeñas tareas que pueden tener duración de un día	<abreviatura>_<Titulo_Funcionalidad _especifica> Ejemplo TS0079_TEMA1_SUBTEMA1 TS0079_TEMA1_SUBTEMA2	1..*

Fuente: elaboración propia, realizado en Microsoft Word.

Cada uno de las propiedades de la plantilla permiten poder tener presente la carga de proyecto si fueran diversos proyectos lo que se llevaran en paralelo en *jira* y a su vez poder crear previamente el desglose de cada *sprint* a trabajar, así también, sus épicas, historias y subtareas para con esta información se pueda crear los elementos necesarios en *jira*.

El crear este listado de elementos permite tener una visualización de todas las posibles tareas que necesitamos realizar para el desarrollo y como se podría distribuir la carga de trabajo en cada *sprint*, además que nos ayuda a ver si existe la dependencia entre diversas historias épicas o tareas con lo cual se puede evitar problemas de dependencias o de atraso en el desarrollo previo a su inicio.

Al tener la plantilla finalizada y distribuida las cargas de trabajo en cada uno de los *sprint* permitira crear un plan de tiempo para el proyecto ya que al conocer las distribuciones y cargas de trabajo por *sprint* se puede estimar el tiempo que conllevara cada tarea y definir si un *sprint* es muy gran o pequeño y adecuarlo acorde a las necesidades.

5.2. Plantilla de *gitlab CI*

La plantilla de *gitlab CI* permitira conocer los elementos necesarios y básicos para poder crear el archivo de configuraciones para la ejecución del *pipeline*. Como primer paso se deben recolectar los datos que se muestran en la siguiente tabla.

Tabla X. Datos necesarios para plantilla

Atributo	Descripción	Nomenclatura
Stages	Agregar el listado de <i>stages</i> que contendrá el proceso	<nombre_stage1>, <nombre_stage2>, <nombre_stageN>
		Ejemplo
		build, test, scan

Continuación de la tabla X.

Jobs	Por cada <i>stage</i> definido en el paso anterior se debe definir una lista de <i>jobs</i> que poseerá	<p><nombre_job1>, <nombre_job2>, <nombre_jobN></p> <p>Ejemplo</p> <p>build_backend build_frontend</p>
Variables	Se debe definir las variables globales del archivo	<p><variable_global1> <variable_global2> <variable_globalN></p> <p>Ejemplo</p> <p>CONTAINER_SONAR_IMAGE CONTAINER_ROBOT_IMAGE</p>
Tags	Se debe listar los <i>tags</i> correspondientes para conocer los ejecutores que utilizaremos	<p><nombre_tag1> <nombre_tag2> <nombre_tagN></p> <p>Ejemplo</p> <p>on-prem rosetta-docker</p>
Artifact	por cada uno de los <i>jobs</i> listados se debe crear un listado de <i>artifact</i> si corresponde	<p><url1> <url2> <urlN></p> <p>Ejemplo</p> <p>frontend-automated-tests/results/*</p>

Fuente: elaboración propia, realizado con Microsoft Word.

Ahora que se conocen los datos que se necesitan recopilar para nuestra plantilla se procede a llenar la tabla con los datos.

Tabla XI. **Datos plantilla**

Atributo	Valor
Stage	<i>build, test, scan, deploy, test-automation, performance</i>
Jobs	<i>build: build_backend, build_frontend test: unit_test_backend scan: sonar_scan deploy: deploy_dev test-automation: test_robot</i>
Variables	CONTAINER_SONAR_IMAGE: "sonar-node:0.1" CONTAINER_SOAPUI_IMAGE: "soapui" CONTAINER_ROBOT_IMAGE: "robot-framework"
Tags	on-prem, rosetta-docker
Artifact	build_backend: webapp-compiler/eirg-webapp/build/libs/*.war build_frontend: webapp/src/build unit_test_backend: api/build/customJacocoReportDir/test/jacocoTestReport.xml api/build/reports/tests/test/* text-automation: frontend-automated-tests/results/*

Fuente: elaboración propia, realizado con Microsoft Word.

Con los valores ya definido en la tabla se puede crear de un archivo de *gitlab-ci* estándar para iniciar el proceso de *DevOps* para cualquier aplicación ya que se tienen todos los elementos necesarios y el cual se puede observar como apéndice 1.

5.3. Plantilla de *ansible*

El poder aprovisionar un servidor de manera automatizada es una de las partes importantes del proceso de *DevOps* y es con el uso de *Ansible* que nosotros realizaremos esta tarea. Es por ello por lo que a continuación se presentan una serie de datos indispensables que se deben conocer para poder armar una plantilla que permita poder entregar los artefactos creados durante las fases anteriores de compilación del *pipeline* a un servidor en el ambiente correspondiente.

Tabla XII. Datos de plantilla *ansible*

Atributo	Descripción	Archivo	Nomenclatura
ambientes	se debe listar los ambientes a los que apuntaremos	<i>host</i>	[<nombre_ambiente> Ejemplo [qa] <host> [dev] <host>
host	direcciones <i>ip</i> de los ambientes	<i>host</i>	<ip> Ejemplo [qa] 192.168.1.2 192.168.1.3
Archivo de keypair del servidor	Se necesita los archivos de clave privada y publica	APP-KEYPAIR APP-KEYPAIR.pub	<nombre>-KEYPAIR <nombre>-KEYPAIR.pub Ejemplo
usuario servidor	Se debe conocer el usuario del	dev/vars.yml	server_user : <usuario> Ejemplo

Continuación de la tabla XII.

	servidor de los ambientes		server_user: weblogic
host_admin	Agregamos la dirección <i>ip</i> de <i>host</i> administrativo del <i>app</i>	dev/vars.yml	admin_server_ip: <ip> Ejemplo admin_server_ip: 192.168.0.1
host_admin_port	Puerto por el cual accederemos al servidor	dev/vars.yml	admin_server_port: <port> Ejemplo admin_server_port: 7001
servers_dev	listado de <i>host</i> y puerto para despliegue	dev/vars.yml	servers_dev: - { name: <ip>, port: <port> } - { name: <ip2>, port: <port2> } Ejemplo - { name: 192.168.1.4, port: 7002 } - { name: 192.168.1.5, port: 7003 }

Fuente: elaboración propia, realizado con Microsoft Word.

Con el listado de los valores que se definen en la anterior tabla se tiene los datos necesarios para poder crear una plantilla estándar que permitiera realizar conexión con los servidores utilizando ansible.

CONCLUSIONES

1. En el análisis del proceso de *DevOps* se crearon una serie de estándares y prácticas que permiten el poder llevar el proceso de una forma ordenada en la implementación como en el proceso de documentación que conlleva.
2. Al recopilar la información necesaria para las plantillas basándose en las tablas de información se puede agilizar el proceso de implementación de los *pipelines*.
3. El poder comparar una serie de herramientas en el mercado que se utilizan en la implementación de cada uno de los procesos de *CI/CD* se obtiene una visión más amplia de lo que sucede en cada proceso y que herramientas pueden ser más útiles dependiendo las necesidades y las demandas que tenga nuestro proyecto.

RECOMENDACIONES

1. Conocer los conceptos básicos del proceso de *DevOps* antes de adentrarse a conocer el proceso completo de implementación.
2. Apoyar en la investigación de otras herramientas para las diferentes etapas de la implementación de un *pipeline* ya que no todas las posibles herramientas que se pueden utilizar en una parte en particular del proceso fueron abordadas.
3. Ahondar en el uso de ciertas herramientas que según nuestro criterio eran las mejores para hacer la implementación estándar del proceso de *DevOps* ya que las plantillas están orientadas a implementarse en dichas herramientas.
4. Conocer conceptos básicos de *cloud computing* para poder comprender que la implementación puede ser llevada a cabo utilizando servicios como los de *Amazon EC2* para creación de instancias en las cuales se pueden crear *runner* o utilizar *Docker* para ejecución y despliegue de las aplicaciones basadas en *react*.

REFERENCIAS

1. Arroyave, H. (2021). *7 alternativas a Jira que compiten con la herramienta de Atlassian*. Recuperado de <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/alternativas-a-jira/>.
2. Atlassian. (2021). *Entrega continua*. Recuperado de <https://www.atlassian.com/es/continuous-delivery>.
3. Aws. (2022). *Operaciones de desarrollo y AWS*. Recuperado de https://aws.amazon.com/es/devops/?nc2=h_ql_sol_use_dops.
4. Azure. (2021). *¿Qué es DevOps?* Recuperado de <https://azure.microsoft.com/es-es/overview/what-is-devops/>.
5. Bala, R. (2021). *Gartner*. Recuperado de <https://cloud.google.com/gartner-cloud-infrastructure-as-a-service/?hl=ES>.
6. Hernández, R. (2018). *12 herramientas imprescindibles para asegurar la calidad del software (y sus alternativas)*. Recuperado de <https://www.genbeta.com/desarrollo/12-herramientas-imprescindibles-para-asegurar-la-calidad-del-software-y-sus-alternativas>.

7. Pittet, S. (2021). *Comparación de integración continua, entrega continua e implementación continua*. Recuperado de <https://www.atlassian.com/es/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment>.
8. Pragma. (2021). *DevOps la cultura ágil para la entrega de software*. Bogotá, Colombia. Recuperado de <https://www.pragma.com.co/academia/conceptos/devops>.
9. Ravooof, S. (2021). *¿Qué es la PaaS? En Qué se Diferencia la Plataforma como Servicio de la IaaS y la SaaS*. Recuperado de <https://kinsta.com/es/blog/que-es-paas/>.
10. React. (2021). *Empezando*. Recuperado de <https://es.reactjs.org/docs/getting-started.html>.
11. Red Hat. (2018). *El concepto de DevOps*. Recuperado de <https://www.redhat.com/es/topics/devops>.
12. Rehkopf, M. (2021). *¿En qué consiste la integración continua?* Recuperado de <https://www.atlassian.com/es/continuous-delivery/continuous-integration>.
13. Schwaber, K y Sutherland, J. (2019). *La Guía de Scrum*. Recuperado de <https://scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-Spanish.pdf#zoom=100>.

14. Sepúlveda, A. (2019). *5 razones por las cuales usar JIRA Portfolio*. Recuperado de <https://www.enevasys.com/5-razones-por-las-cuales-usar-jira-portfolio/>.
15. Sonarqube. (2022). *SonarQube Documentation*. Recuperado de <https://docs.sonarqube.org/latest/>.

APÉNDICE

Apéndice 1. Archivo.gitlab-ci.yml de ejemplo

```
variables:
  CONTAINER_SONAR_IMAGE: "sonar-node:0.1"
  CONTAINER_SOAPUI_IMAGE: "soapui"
  CONTAINER_ROBOT_IMAGE: "robot-framework"

stages:
  - build
  - test
  - scan
  - deploy
  - test-automation
  - performance

build_frontend:
  stage: build
  script:
    - cd webapp-compiler
    - sudo gradle clean build -x test --configure-on-demand --daemon --parallel
  tags:
    - on-prem
  artifacts:
    paths:
      - webapp-compiler/eirg-webapp/build/libs/*.war

build_backend:
  stage: build
  image: gradle:6.6.1-jdk8
  script:
    - cd ear
    - gradle clean build -x test --configure-on-demand --daemon --parallel
  tags:
    - on-prem
  artifacts:
```

Continuación del apéndice 1.

```
paths:
  - webapp-compiler/eirg-webapp/build/libs/*.war

unit-test-backend:
  stage: test
  image: gradle:6.6.1-jdk8
  script:
    - cd webapp-compiler
    - gradle test
  tags:
    - on-prem
  artifacts:
    paths:
      - api/build/customJacocoReportDir/test/jacocoTestReport.xml
      - api/build/reports/tests/test/*

sonar-scan:
  image: ${CI_REGISTRY_IMAGE}/${CONTAINER_SONAR_IMAGE}
  stage: scan
  script:
    - chmod +x sonarscan.sh
    - ./sonarscan.sh
  tags:
    - rosetta-docker
  dependencies:
    - unit-test-backend

front-end-tests-dev:
  stage: test-automation
  image: ${CI_REGISTRY_IMAGE}/${CONTAINER_ROBOT_IMAGE}
  before_script:
    - python3 -m venv env
    - source ./env/bin/activate
    - python -m pip install robotframework-selenium2library
    - python -m pip install --upgrade pip
  script:
    - cd eirg-frontend-automated-tests
    - robot -d results -i TestPlan1 .
  tags:
```

Continuación del apéndice 1.

- on-prem
artifacts:
paths:
- eirg-frontend-automated-tests/results/*
only:
- develop
when: manual

Fuente: elaboración propia, realizado con Microsoft Word.