Implementación de una arquitectura de integración y entrega continua basada en contenedores y buenas prácticas DevOps en entornos de desarrollo web para la empresa Emagic S.A

Cutiopala Morocho, Luis David y Taday Leon, Kevin Alexander

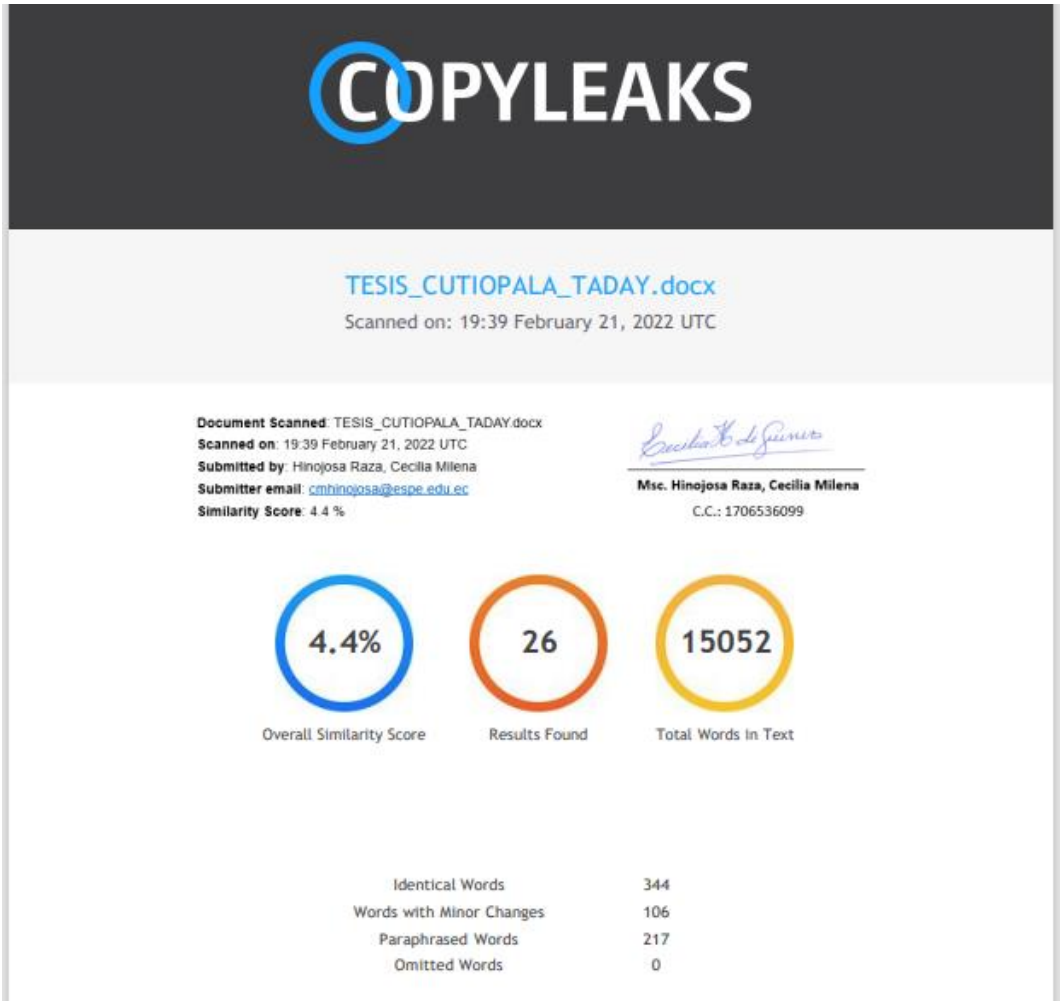
Departamento de Ciencias de la Computación

Carrera de Ingeniería de Sistemas e Informática

Trabajo de titulación, previo a la obtención del título de Ingeniero en Sistemas e Informática

Msc. Hinojosa Raza, Cecilia Milena

20 de octubre del 2021



COPYLEAKS

TESIS_CUTIOPALA_TADAY.docx

Scanned on: 19:39 February 21, 2022 UTC

Document Scanned: TESIS_CUTIOPALA_TADAY.docx
Scanned on: 19:39 February 21, 2022 UTC
Submitted by: Hinojosa Raza, Cecilia Milena
Submitter email: cmhinojosa@espe.edu.ec
Similarity Score: 4.4 %

Msc. Hinojosa Raza, Cecilia Milena
C.C.: 1706536099

4.4% Overall Similarity Score

26 Results Found

15052 Total Words in Text

Identical Words	344
Words with Minor Changes	106
Paraphrased Words	217
Omitted Words	0

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

CERTIFICACIÓN

Certifico que el trabajo de titulación, "**Implementación de una arquitectura de integración y entrega continua basada en contenedores y buenas prácticas DevOps en entornos de desarrollo web para la empresa Emagic S.A**" fue realizado por los señores **Cutiopala Morocho Luis David** y **Taday Leon Kevin Alexander**, el cual ha sido revisado y analizado en si totalidad por la herramienta de verificación de similitud de contenido; por lo tanto cumple con los requisitos legales, teóricos, científicos, técnicos y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, razón por la cual me permito acreditar y autorizar para que lo sustente públicamente.

Quito, 26 de octubre de 2021

Msc. Hinojosa Raza, Cecilia Milena

CC: 1706536099

UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

RESPONSABILIDAD DE AUTORÍA

Nosotros **Cutiopala Morocho Luis David** con C.C.: 1723305502 y **Taday Leon Kevin Alexander** con C.C.: 1726213976, declaramos que el contenido, ideas y criterios del trabajo de titulación: **Implementación de una arquitectura de integración y entrega continua basada en contenedores y buenas prácticas DevOps en entornos de desarrollo web para la empresa Emagic S.A,** es de nuestra autoría y responsabilidad, cumpliendo con los requisitos legales, teóricos, científicos, técnicos y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, respetando los derechos intelectuales de terceros y referenciando las citas bibliográficas.

Quito, 26 de octubre de 2021

Cutiopala Morocho Luis David
C.C.: 1723305502

Taday Leon Kevin Alexander
C.C.: 1726213976

UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

AUTORIZACIÓN DE PUBLICACIÓN

Nosotros **Cutiopala Morocho Luis David** con C.C.: 1723305502 y **Taday Leon Kevin Alexander** con C.C.: 1726213976, autorizamos a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de titulación: **Implementación de una arquitectura de integración y entrega continua basada en contenedores y buenas prácticas DevOps en entornos de desarrollo web para la empresa Emagic S.A,** en el Repositorio Institucional, cuyo contenido, ideas y criterios son de nuestra responsabilidad.

Quito, 26 de octubre de 2021

Cutiopala Morocho Luis David

C.C.: 1723305502

Taday Leon Kevin Alexander

C.C.: 1726213976

Dedicatoria

David

Para mis padres, Daniel, Nicole, Sofia y amigos que en su momento me ayudaron en este largo camino.

Kevin

Para mí, que todos estos años de esfuerzo han ayudado a forjar más mi carácter, y entender que los obstáculos que aparecieron no fue más que desafíos para superarme a mí mismo.

Agradecimiento

David

Agradezco a mi familia, amigos, profesores y la Msc. Cecilia Hinojosa quien con su sabiduría y experiencia nos ha guiado y que me ha impartido su conocimiento.

Kevin

Agradezco a mi familia, amigos, profesores y la Msc. Cecilia Hinojosa quien con su sabiduría y experiencia nos ha guiado y que me ha impartido su conocimiento.

Índice de Contenido

Índice de Tablas

Índice de Figuras

Resumen

En el mundo actual, dos factores inciden en la calidad del software, por un lado, la complejidad de los requerimientos y por otro, las prácticas que aplica la industria del software. Ante esta situación, el presente trabajo de titulación define los procesos y flujo de trabajo para la implementación de una arquitectura basada en contenedores y buenas prácticas DevOps, con el fin de entregar productos de alta calidad en el tiempo previsto y con el presupuesto planificado.

El documento se enfoca en el uso de la metodología Scrum, la utilización de contenedores y buenas prácticas DevOps que se enlazan entre sí para obtener un mejor rendimiento en el despliegue de aplicaciones, obteniendo así una mejora considerable en el código y la comunicación entre los involucrados en este proceso.

Adicionalmente se indica como se integran las diferentes herramientas de integración y entrega continua las cuales son necesarias para cumplir con todo el ciclo DevOps.

Para validar la eficiencia de la arquitectura implementada se realizó un análisis comparativo entre la arquitectura que viene aplicando la empresa auspiciante y la nueva arquitectura propuesta; para ello se utilizaron diferentes indicadores y variables que permiten determinar los beneficios que presenta la nueva arquitectura un resultado considerablemente bueno en cuanto a rendimiento y mejora de tiempos para realizar la integración y entrega continua.

Palabras clave:

- **DEVOPS**
- **MICROSERVICIO**
- **INTEGRACIÓN CONTINUA**
- **ENTREGA CONTINUA**

Abstract

The current job creation defines the implementation of a continuous integration and integration architecture based on DevOps containers and good practices.

This document is derived from the use of the Scrum methodology, the use of DevOps content and good practices that are included in order to obtain a greater return from the application, obtaining a significant increase in the number of entries and the comma. It is a process.

This work indicates how the different integration tools are integrated and continuous, the numbers are necessary to complete the entire DevOps cycle.

To support the implemented architecture, an analysis is carried out between the architecture managed by the company and the new proposed architecture, in order to use different indicators and variables that measure the difference between the measures, obtaining a considerably better result than the times to be measured. achieve continuous integration and input.

Key words:

- **DEVOPS**
- **PIPELINE**
- **CONTINUOS INTEGRATION**
- **CONTINUOS DELIVERY**

Capítulo I

Introducción

Antecedentes

No solo la tecnología mejora a pasos agigantados, sino que la manera en que las compañías se organizan también ha evolucionado, requiriendo con ello nuevas maneras de comunicación entre los empleados de las diferentes áreas. Las empresas siempre se enfocan en cumplir su objetivo principal, que será ofrecer un producto o servicio de calidad.

El brindar servicios tecnológicos que suministren a los clientes asistencia a los procesos del negocio se ha transformado en uno de los nuevos objetivos de las empresas que investigan la optimización en sus procesos o tareas.

La ejecución de una planificación donde de como resultados sobrepaso de tiempos da como resultado a la molestia de clientes, se sentirán insatisfechos y molestos, a medida que siga sucediendo esto, la empresa va a sufrir perdidas económicas.

A raíz de esto, surgió entre los profesionales un movimiento para tratar de romper las barreras que separaban a los departamentos de desarrollo y operaciones, el movimiento Development and Operations (DevOps).

El glosario de tecnologías de información (Gartner, 2017) define DevOps como *"La transformación sobre la cultura de TI, dando como uno de sus objetivos la prestación eficaz y rápida de los servicios de TI mediante la adopción de prácticas ágiles en el entorno de TI, que da un enfoque orientado al sistema. DevOps hace empeño en el personal y todas las personas involucradas, ya que busca mejorar la colaboración entre los equipos de desarrollo y las operaciones. DevOps utiliza tecnologías innovadoras, sobre todo herramientas de automatización que pueden servir para una infraestructura cada vez más dinámica y programable desde la perspectiva del ciclo de vida."*

Durante los últimos años se han desarrollado nuevas herramientas que han cambiado la forma en la que los administradores de sistemas y desarrolladores colaboran para producir software de mejor calidad.

En la investigación de (Forsgren, 2018) llega a determinar que la adopción de DevOps es clave para las organizaciones de alto rendimiento (Netflix, Amazon, Google, and Facebook), tuvieron: 46 veces más frecuencia de despliegue de código, tiempo de entrega de 440 veces más rápido desde el compromiso hasta la implementación, un tiempo medio para recuperación (MTTR) de 170 veces más rápido, tasa de falla de cambio 5 veces menor (1/5 de probabilidad de que falle un cambio). En Perú, el Banco BBVA en el año 2016 inició la implementación de DevOps para acortar el ciclo de desarrollo de nuevas aplicaciones, en el 2018 ya contaba con una mejora del 45% en el tiempo de desarrollo de las nuevas aplicaciones adoptando DevOps (BBVA, 2018).

Emagic S.A es una empresa que inicio las operaciones en el año 2006, dedicado al desarrollo tecnológico y a la innovación, en la actualidad, esta denominada como “Auxiliar de Servicios Financieros” que ofrece soluciones de software, tanto para las entidades bancarias privadas y públicas.

Uno de los objetivos de la empresa es crear productos que satisfacen la necesidades del cliente y entregar productos de calidad, procurando mantener la disponibilidad, integridad y confidencialidad de la información, tanto interna como externa, generando las mejores prácticas para que el personal administre y resguarde la información de forma segura, garantizando la reputación de la empresa y la continuidad del negocio.

Planteamiento del Problema

Emagic S.A. desarrolla aplicaciones web que se despliegan sobre sistemas operativos Linux. En la actualidad la empresa basa sus proyectos en Centos 7, una distribución ideal para su uso en servidores.

Sin embargo, con el paso del tiempo, el desarrollo de las distintas aplicaciones requiere de componentes más complejos que necesitan de una o más configuraciones manuales por parte del administrador de sistemas, estas configuraciones a su vez deben ser replicadas en los ordenadores locales de los desarrolladores involucrados, sin embargo, estas configuraciones no son realizadas eficientemente y en ocasiones generan conflictos en el momento que son desplegadas en ambientes de producción.

La configuración para un nuevo aplicativo, a la empresa le lleva en promedio 48 horas, debido a lo complicado que es la creación de un entorno de un proyecto web, ya que estos necesitan varios componentes como un servidor web, un servidor de base de datos, o un servidor de correo entre otros. Adicionalmente, cuando se integra personal nuevo a la empresa, el proceso de capacitación para que pueda ejecutar estas tareas dura 40 horas, lo cual genera pérdida de tiempo.

Adicionalmente, según informes entregados por la empresa Emagic S.A. se menciona que, para la implementación de un software, que va a ser desplegado en un ambiente de producción, este proceso llega a demorar hasta 48 horas máximo, debido a la capacidad de los ambientes no productivos, pruebas manuales y gestión de cambios no llevados correctamente.

Los problemas mencionados anteriormente generan retrasos en la entrega final del producto, lo que incrementa los costos de producción y genera insatisfacción del cliente.

Justificación

La presente investigación se enfocará en implementar una arquitectura que ayude a la integración y entrega continua de aplicaciones web, ya que debido a los recientes estudios

realizados la empresa Emagic S.A, presenta errores, defectos y problemas cíclicos que hacen que el proceso de integración y entrega continua no se cumpla correctamente, en la actualidad los departamentos de desarrollo e infraestructura tienen desfases a la hora trabajar conjuntamente, por un lado el desarrollo de un software conlleva procesos que retrasan su lanzamiento en ambientes de producción, este aislamiento entre los departamentos mencionadas genera: reprocesos, demoras en las entregas, baja calidad y poca estabilidad del software.

Así, el presente trabajo permitirá mostrar los cambios entre la arquitectura tradicional manejada por la empresa y la arquitectura propuesta, ayudando a obtener mejoras en el tiempo de entrega, calidad, seguridad y la estabilidad del software, generando valor a la empresa. Con la implementación propuesta a más de contribuir con una solución al problema ya descrito, también se puede contribuir a la industria del desarrollo de software en general.

Objetivos

Objetivo General

Implementar una arquitectura basada en contenedores y buenas prácticas DevOps en entornos de desarrollo web para la integración y entrega continua de aplicaciones web en la empresa Emagic S.A.

Objetivos Específicos

- Realizar un estudio de literatura para identificar los métodos y tecnologías como base, para el desarrollo del proyecto en la empresa Emagic S.A.
- Definir los procesos de construcción de la arquitectura que se ajuste a las necesidades y presupuesto de la empresa, estos incluyen la construcción de las

aplicaciones, test unitarios, y el despliegue en contenedores.

- Validar el proceso y la arquitectura implementada a través de pruebas continuas y análisis de información sobre satisfacción del usuario.

Alcance

Este estudio comprende la implementación de una arquitectura basada en contenedores bajo buenas prácticas y una Cultura DevOps para la integración y entrega continua de los productos de software.

Se implementará la propuesta en la empresa Emagic S.A, el cual posee una cartera de productos y servicios para entidades bancarias públicas y privadas.

El alcance del presente proyecto de investigación se ha divido en varias fases para mejorar su comprensión:

- Obtener información de herramientas que ayuden al desarrollo del proyecto de investigación.
- Implementar y validar la arquitectura propuesta en un prototipo de la empresa Emagic S.A
- Realizar un estudio comparativo de la arquitectura que maneja actualmente la empresa con la arquitectura propuesta.
- Evaluar y monitorear la arquitectura basada en la metodología ágil SCRUM y buenas prácticas DevOps con el fin de construir un software menos propenso a errores de incompatibilidad.

Con el fin de validar la arquitectura propuesta, realizará la implementación, en donde se obtendrán datos para analizarlos cuantitativa y cualitativamente, para sus posibles implementaciones en organizaciones similares del país.

En la Tabla 1 se describe con mayor detalle el alcance del presente proyecto, para el efecto se presentan varias preguntas de investigación que se encuentran relacionadas con cada uno de los objetivos específicos.

Tabla 1.

Objetivos y preguntas de investigación

Objetivo Específico	Pregunta de Investigación
i. OE1. Realizar un estudio de literatura para identificar los métodos, técnicas y tecnologías que ayuden alcanzar la arquitectura de integración y entrega continua en la empresa Emagic S.A.	a. ¿Qué incidencia tiene el análisis de los indicadores de desempeño con el éxito o fracaso de proyectos?
ii.OE2. Definir los procesos de construcción de la arquitectura que se ajuste a las necesidades y presupuesto de la empresa estos incluyen la compilación de las aplicaciones, test unitarios, y despliegue de los ambientes en contenedores.	b. ¿Cuáles son las metodologías y herramientas más adecuadas para la implementación de DevOps? c. ¿Cuál es el proceso que se realiza al aplicar integración y entrega continua en aplicaciones web?

iii.OE3. Validar que el proceso y la arquitectura implementada cumpla con los objetivos mediante entrevistas semiestructuradas con los interesados.	d. ¿Cómo se llega a validar que la arquitectura propuesta cumpla con los objetivos esperados?

Estado del arte

Planteamiento de la Revisión Sistemática.

En esta fase se efectuó una descripción breve del problema de investigación que provee un contexto para la exploración de estudios científicos; consecutivamente se procedió a precisar un objetivo de búsqueda y plantear preguntas de investigación, para organizar la búsqueda en dependencia al problema de investigación y posteriormente se definieron los criterios de inclusión y exclusión.

Criterios de Inclusión

- Estudios enfocados en el análisis y gestión de indicadores de desempeño en procesos de la investigación.
- Estudios que hablen de los factores relacionados con la falta de innovación en las universidades.
- Estudios que presenten propuestas para mejorar la innovación en las universidades.
- Estudios enfocados en estrategias para implementar y desarrollar proyectos de innovación.

Criterios de Exclusión

- Estudios que hablen de la innovación en general y no centrados en el ámbito universitario

o de la investigación.

- Artículos que se encuentren redactados en otro idioma que no sea inglés.
- Artículos publicados antes del año 2012 o publicados en revistas no muy relevantes.

Definición del grupo de control y extracción de términos.

Según (Petersen, Feldt, Mujtaba, & Mattsson, 2008) es fundamental establecer la bibliografía para formar los artículos que sean importantes para la investigación, y eliminando aquellos que solo mencionan un enfoque generalizado.

Es importante la integración de al menos dos o tres investigadores. Cada investigador propone estudios e investigaciones que podrían ser parte del grupo de control. Se incluyo una validación cruzada donde se estableció el grupo de control, y se detalla en la Tabla 2.

Tabla 2.

Conformación grupo de control

Título	Cita	Palabras clave
Fast Delivery, Continuously Build, Testing and Deployment with DevOps Pipeline Techniques on Cloud	(Muhammad Owais Khan., 2020)	DevOps Pipeline, Culture and CI (Continuous Integration), CD (Continuous Delivery) Implementation.
A New Agile Framework that supports DevOps, Continuous Delivery, and Lean Hypothesis Testing.	(Jeffrey Saltz., 2020)	CI (Continuous Integration), CD (Continuous Delivery), Implementation on Cloud
Implementation of a continuous delivery pipeline for Enterprise architecture model Evolution	(Alex R. Sabau, 2020)	Continuous delivery

Construcción de la Cadena de Búsqueda

Se conformo la cadena de búsqueda con las palabras clave que fueron obtenidas de los artículos científicos:

TITLE-ABS-KEY ("CI" OR ("integration*" and "continuous")) AND ("CD" OR ("deliver*"and"continuous")) and " and "devops"), misma que se utilizó en la base digital SpringerLink.

Sin embargo, se consiguió un gran número de investigaciones con esta cadena, incluso aquellas que fueron retiradas. Posteriormente se realizó varias pruebas con distintas composiciones de cadenas, y se seleccionó la cadena:("CI" OR ("integration*" and "continuous")) AND ("CD" OR ("deliver*"and"continuous")) and "open source" and "tool*" and "DevOps"and “containers”

Selección de los Estudios Primarios

Al emplear la cadena de búsqueda en la base digital de SpringerLink se consiguió alrededor de 62 artículos afines con el tema y se consideró un número de artículos posibles de manejar; adicionalmente con esta cadena la mayor parte de los artículos del grupo de control apareció dentro de los artículos encontrados.

Se aplican 2 filtros de los 62 artículos encontrados, se detallan a continuación:

- **Vigencia:** Estudios realizados a partir del año 2018. Este año es elegido ya que la tecnología avanza a pasos acelerados, por lo que es relevante tener estudios con una relativa actualidad.

- **Tipo de estudio:** Se eligieron únicamente estudios del tipo: technical report, conference paper y journal paper; debido a su relevancia.

En base a los filtros antes mencionados, y el criterio de los investigadores, se eligieron 3 estudios primarios, los cuales constituyen la base para realizar el estudio del estado del arte, los cuales se muestran en la tabla 3.

Tabla 3.

Estudios Seleccionados

Código	Título	Cita
EP1	Enabling continuous integration in a formal methods setting	(Luis Diogo Couto 1, 2019)
EP2	Continuous integration for laravel applications with Gitlab	(Farhana Sethil., November 2019)
EP3	DevOps in practice: A multiple case study of five companies	(Lucy Ellen Lwakatare, 2019)

Elaboración de Estado de Arte

- **EP1 (Luis Diogo Couto1, 2019): Enabling continuous integration in a formal method setting**

En el desarrollo de software moderno, las prácticas de integración continua y DevOps se utilizan ampliamente para aumentar la entrega. acelerar y reducir el tiempo que lleva implementar cambios de software en producción. Si las herramientas del método formal no pueden ser integradas en un paradigma DevOps, entonces se reducirá su impacto en el desarrollo de software. En este artículo se presentan trabajos para abordar este problema a través de una serie de extensiones para la herramienta Overture que respalda el Método de Desarrollo de Viena. Las extensiones permiten que Overture se utilice en un entorno DevOps, a través de la

integración y validación continua de modelos y código generado a través de la integración con el servidor de automatización de Jenkins. Enmarcan la integración de métodos formales y DevOps en una serie de principios, demuestran el valor de esta integración a través de un estudio de caso y reflexionan sobre las experiencias utilizando métodos formales y DevOps en un entorno industrial. Los autores esperan que este trabajo pueda ayudar a otros practicantes de métodos formales integrar sus herramientas con DevOps.

- **EP2 (Meng et al., 2016): Continuous integration for applications with Gitlab**

Las industrias modernas de desarrollo de software están dispuestas a utilizar la integración continua (CI) en lugar de la integración tradicional para sus cambiantes requisitos comerciales. Los desarrolladores enfrentan desafíos en procesos manuales como revisión de código, combinación de código y prueba de código cuando integran su código con frecuencia. CI es una de las prácticas ágiles más populares y comúnmente aspiradas.

- **EP3 (Lucy Ellen Lwakatare, 2019): DevOps in practice: A multiple case study of five companies**

DevOps, un acrónimo de desarrollo y operaciones es un enfoque donde los desarrolladores de software y las operaciones trabajan en estrecha colaboración. El objetivo es mejorar la comunicación y la integración del desarrollo y las operaciones para aprovechar al máximo los beneficios del software moderno. enfoques de desarrollo que emplean lanzamientos rápidos de nuevo software características para los usuarios finales y, posteriormente, aprender de ellos.

Capítulo II

Marco Teórico

Ingeniería de Software

Es un área de las Ciencias de la Computación, que brinda técnicas y metodologías para construir y sostener software eficiente y de calidad que solucionen problemas de cualquier tipo del mundo real.

Las aplicaciones y sistemas basados en web iniciaron como un simple conjunto o vínculos de contenidos de información a sistemas de TI refinados con funcionalidades complejas y contenidos en varios medios y/o dispositivos electrónicos.

A inicio de los años (1990 – 1995), los sitios web constituían en una serie de archivos de hipertexto vinculados, que mostraban imágenes y texto de manera reducida.

Con el tiempo y con la mejora del HTML se evoluciono a mecanismos de desarrollo tales como XML que es una forma de transmitir información o Java que es un lenguaje de programación, esto brindó a los especialistas de este campo, a la capacidad amplia de cómputo que se podría procesar, así como a la transferencia de información.

“Las aplicaciones web pertenecen a una categoría de software, entre publicaciones de imágenes o texto y desarrollo de software, entre la computación y el marketing y entre la tecnología y el arte” (Pressman, 2015).

Características de las aplicaciones web

- Utilización de protocolos de internet y redes, un aplicativo web se aloja en una red y brinda requerimientos de diversos clientes, así dando acceso a la comunicación.

- Concurrencia, es la capacidad de respuesta ante las solicitudes de los clientes hacia el sitio web, momentos o situaciones en que los usuarios utilizan al mismo tiempo el sitio web.
- Acceso a los datos, por lo general las aplicaciones web son manejadas para acceder a la información que persiste en repositorios de datos que atañen al ambiente global web (aplicaciones financieras, comercio electrónico).
- Rapidez, poseer herramientas nuevas y estables que hace posible la construcción de aplicaciones web, en menor tiempo.

Modelado de requerimiento para aplicaciones web

Según (Pressman, 2015) el grado de profundidad de la configuración de los requerimientos, depende de los siguientes factores:

- Tamaño y complejidad del aumento de aplicaciones web.
- Identificar y establecer los requerimientos con el fin de determinar el número de personas involucradas en el equipo de desarrollo.

Modelo funcional para las aplicaciones web

Según (Pressman, 2015) el modelo funcional afronta 2 elementos de procesamiento, entre los cuales:

1. Funciones visibles por el usuario, es la consecuencia visible de una función efectuada por la empresa y que pone a disponibilidad del cliente final.
2. Operaciones contenidas en las clases de análisis, la mayor parte del problema de los aplicativos no habita en las funcionalidades que abarcan sino en la naturaleza de la información a la que se accede.

Arquitectura de Software

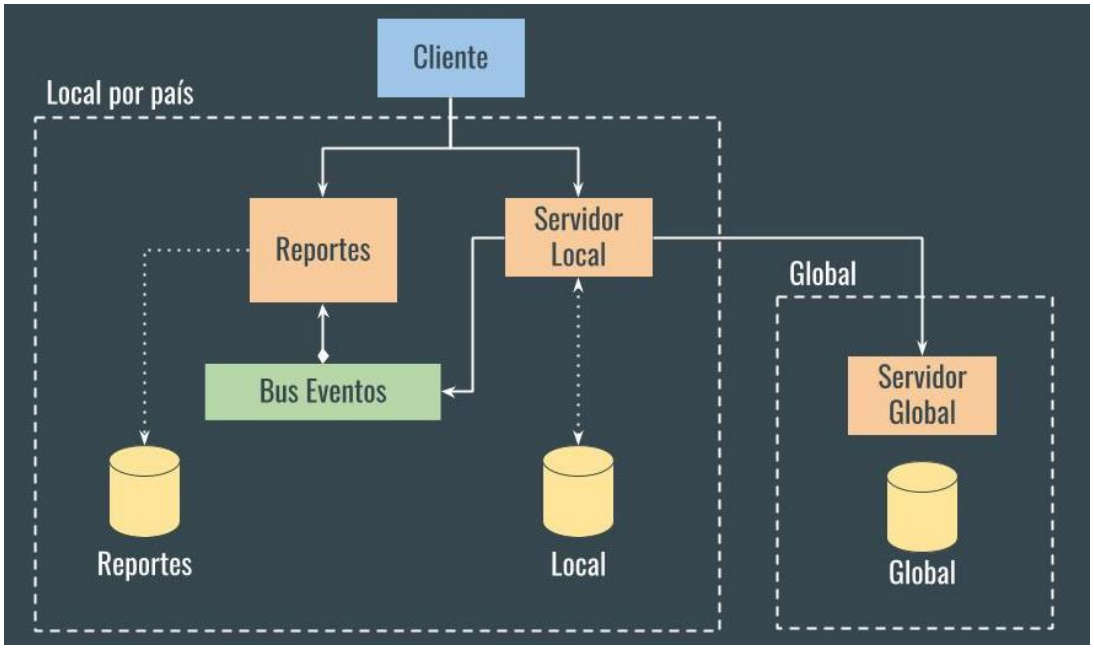
La Arquitectura de Software representa y establece el modelo a construir para definir la estructura de una aplicación de software. Este está conformado por los paradigmas, diseños, patrones de arquitectura y políticas que se aplican en el diseño y posterior implementación. Todo esto alineado con los atributos de calidad como el rendimiento, la seguridad, usabilidad y confiabilidad.

“La arquitectura engloba los conceptos y atributos fundamentales en el ámbito de un sistema, en los cuales son representados en cada parte por vínculos y en los principios que determinan su avance y diseño.” (ISO/IEC/IEEE, 2016)

Con lo anteriormente mencionado, contribuye a definir la estructura del sistema a través de esquemas que dan guía a la construcción del software. En la actualidad se puede prestar atención a diversos conceptos de arquitectura de software. Gracias a esto se puede representar la arquitectura de software en diagramas como se puede ver en la ilustración 1.

Figura 1.

Descripción de Arquitectura de un estilo específico



Nota: Adaptado de (Platzi, 2018)

Diseño de la Arquitectura

Uno de los puntos más importantes de la Arquitectura de Software es seleccionar los estilos y patrones apropiados que den el soporte requerido y necesario para cumplir con las propiedades de calidad deseados.

Estilos de Arquitectura

Hoy en día coexisten un conjunto de tesis para los diferentes estilos arquitectónicos de software; según la definición dada por (Campos, 2018), nos dice que un estilo es "*Una familia de sistemas en términos de patrón de organización estructural. Concretamente, un estilo arquitectónico establece los conectores y componentes que se puede usar en un aplicativo, adicional a un conjunto de reglas de cómo pueden ser complementadas*".

Cabe recalcar que (Gutierrez, 2015) menciona que cada estilo explica una clase de sistemas, que comprende:

- Grupo de componentes, como: módulos computacionales y BD, que desempeñan una función explicita para el funcionamiento global del sistema.
- Limitaciones que definen cómo se forman los componentes para crear el sistema.

En la Tabla 4 se citan las principales áreas de enfoque y los estilos arquitectónicos correspondientes:

Tabla 4.

Estilos Arquitectónicos de Software

Categoría	Estilos de Arquitectura
Comunicación	Arquitectura orientada a servicios (SOA)
Despliegue	Cliente / Servidor, N-Tier
Estructura	Arquitectura basada en componentes, orientada a objetos y en capas

Fuente: Propia

Cabe recalcar que los objetivos de la empresa a corto y largo plazo, la capacidad que posee para diseñar, implementar y monitorear su infraestructura, da resultado a formar sus componentes de gran dominio al momento de elegir el estilo arquitectónico que mejor se adapte y necesite.

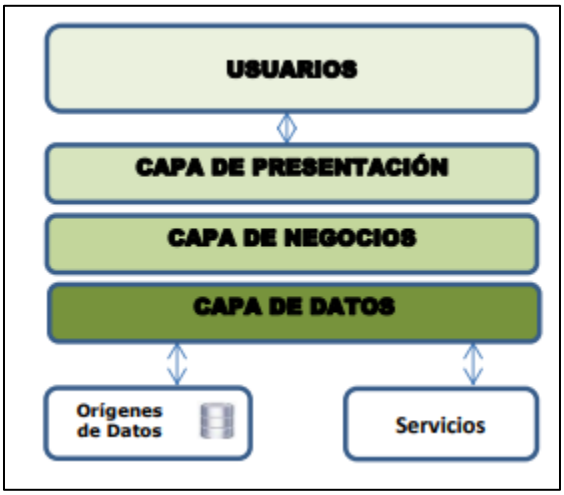
Arquitectura en capas o monolítica

La arquitectura en capas se brinda un conjunto de servicios a las capas superiores y de los cuales demanda servicios de las inferiores según se necesario. Ver ilustración 2.

"*La arquitectura establecida por capas enfatiza en la gestión de roles y responsabilidades de forma jerárquica dando a una forma muy efectiva de independencia de responsabilidades u obligaciones."* (Microsoft, 2019)

Figura 2.

Componentes de la arquitectura en capas



Nota: Propia

De acuerdo con la ilustración 2 presentada, es concerniente indicar que la arquitectura en capas es constituida por algunos componentes (capas) como son:

- Componentes de interfaz de usuario (UI), que forma la capa de presentación.
- Componentes de lógica del negocio, que hace referencia a las interfaces de servicios o controladores, que estas involucran a las entidades empresariales y sus respectivos flujos de trabajo.
- Componentes de acceso a datos, en esta capa se accede al origen de los datos a través de consultas SQL (Structured query languaje).

Ventajas:

- Alta cohesión y bajo acoplamiento
- Admite la estandarización de servicios.
- Brinda una gran reutilización de código o servicios, que estas pueden contribuirse entre sí, gracias a que cada interfaz es lo suficientemente clara.
- Ayudan al mantenimiento, ya que los cambios solo afectan a las capas contiguas.

Arquitectura Orientada a Servicios

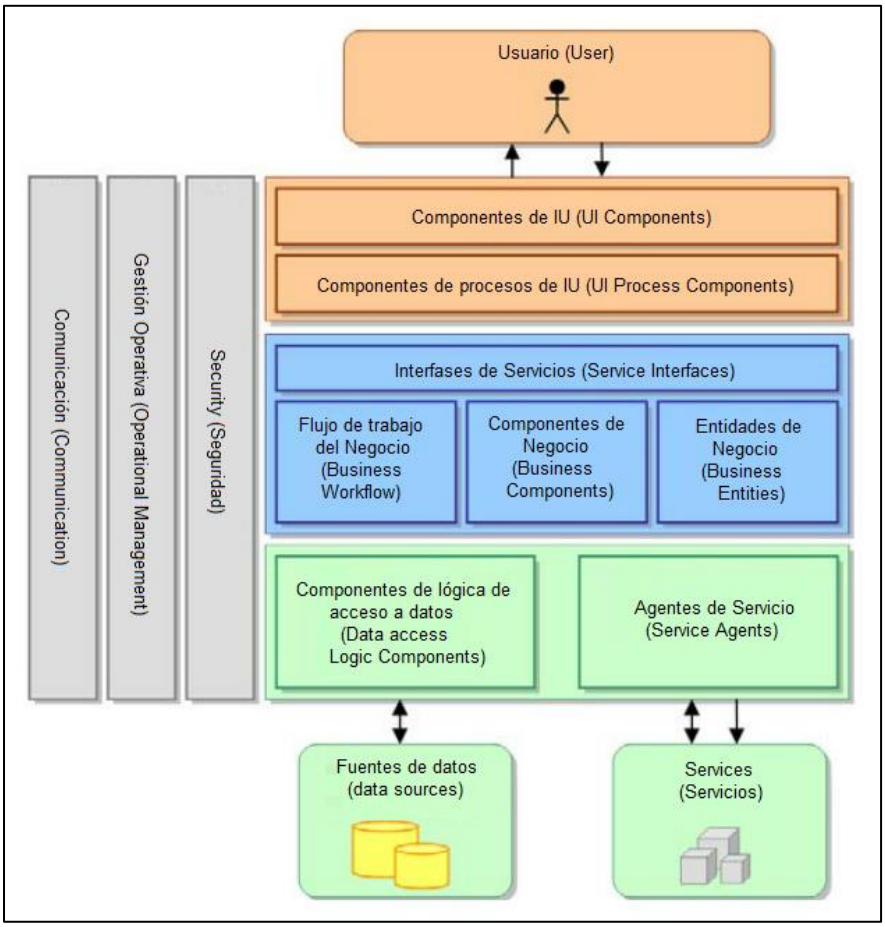
La Arquitectura Orientada a Servicios (SOA) se define como una orientación de diseño que permite crear aplicaciones de negocio, en las cuales sus funcionalidades son expuestas como servicios hacia al mundo exterior, y que se pueden acceder en forma independiente y remota usando interfaces estandarizadas. Microsoft lo cita de la siguiente manera:

"*La arquitectura orientada a servicios, conocida por sus siglas SOA, hace hincapié a una perspectiva de mejora de las aplicaciones de software empresarial, donde sus procesos se dividen en servicios, los cuales son visualizados a través de una red.*" (Microsoft Corporation, 2016)

La publicación de (Bocchio, 2015) muestra una vista lógica para referenciar la arquitectura en donde las instancias se pueden desarrollar para una plataforma y una tecnología específica. En la ilustración 3 se presenta esta vista lógica.

Figura 3.

Vista Lógica de la Arquitectura SOA



Nota: Adaptado de (Bocchio, 2015)

A continuación, se describe cada capa con su definición:

- **Capa de Sistemas Operacionales**. Esta capa adiciona el sistema operativo en el cual se ejecuta o ejecutará el entorno de TI de la empresa que son utilizados, incluyen las aplicaciones, sistemas de procesamiento de transacciones y las diversas fuentes de origen de la información.
- **Capa de Componentes de servicios**. Esta capa garantiza la calidad de servicio (QoS) y el desempeño.

- **Capa de servicios.** Se encuentra los servicios definidos por la empresa, que estas compuestos en su carácter sintáctico (operaciones de servicios, la entrada y salida de mensajes, definición de fallos) y la información a transmitir.
- **Capa de procesos de Negocio**. La lógica de cada proceso que constituye al negocio, además de describir cómo funcionan, con el objetivo de cumplir con las premisas del negocio.
- **Capa de Consumo**. Se encuentran en canales donde pueden ser consumido de manera externa e interna, y que acceden a la funcionalidad de la aplicación.

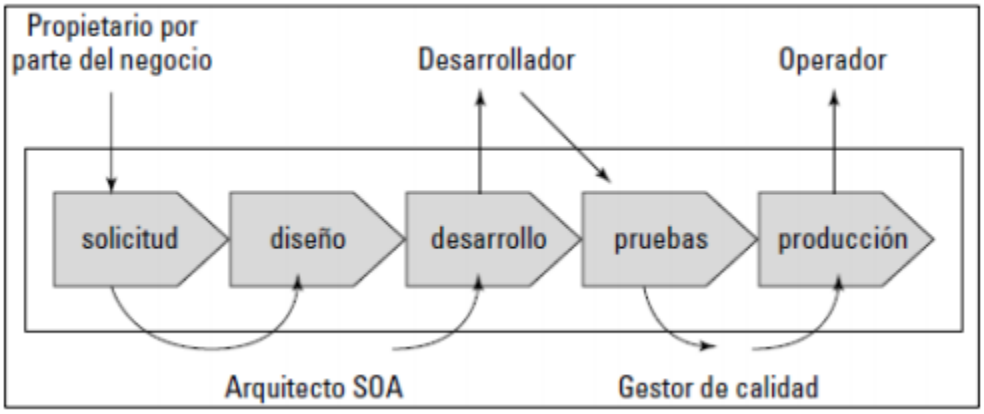
Ciclo de Vida de un Servicio.

En el gobierno de SOA, el ciclo de vida guía el trabajo de la mano con los procesos de la gestión de proyectos en colaboración con los arquitectos y el equipo de TI, dando así al cumplimiento de las reglas y políticas.

En la Ilustración 4 se puede observar el ciclo de vida de un Servicio.

Figura 4.

Ciclo de vida de un Servicio



Nota: Adaptado de (Matsumura, Brauel, & Shah, 2009)

Principios SOA.

Los principios sirven a través de reglas o políticas que son empleadas por los equipos de trabajo para efectuar o mantener una Arquitectura que esté basada en este tipo.

El libro Principios de Diseño de Servicios por (Erl, 2014) indican los siguientes principios:

- Bajo Acoplamiento
- Reusabilidad
- Autonomía
- Combinación

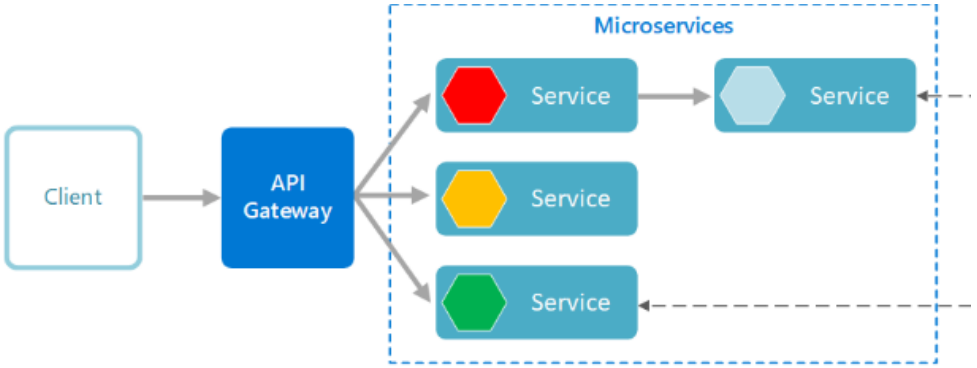
Arquitectura de Microservicios

Una arquitectura de microservicios es un tipo de arquitectura basado en pequeños servicios que pertenecen un área de negocio específica, los cuales son independientes y autónomos, y constituyen una intercomunicación entre ellos mediante peticiones y protocolos HTTP.

Cada microservicio es independiente y puede ser desplegado sin afectar a los demás microservicios, ya que solo exponen la API al mundo exterior. En la Ilustración 5 se representa presenta la arquitectura y como cada microservicio es expuesto.

Figura 5.

Arquitectura de Microservicios



Nota: (Microsoft Corporation, 2019)

Según (Fowler, 2014) "*El estilo arquitectónico de un microservicio, es un enfoque para desarrollar una sola aplicación como un conjunto de pequeños servicios, cada uno ejecutándose en su propio proceso y comunicándose con mecanismos ligeros, a menudo una API de recursos HTTP. Estos servicios se basan en capacidades empresariales y se pueden implementar de forma independiente mediante maquinaria de despliegue completamente automatizada. Existe un mínimo de gestión centralizada de estos servicios, que puede escribirse en diferentes lenguajes de programación y utilizar diferentes tecnologías de almacenamiento de datos.*"

Entre las ventajas que se pueden obtener son:

- El despliegue es independiente y modular.
- Su comprensión es fácil debido a que la lógica del negocio se localiza bien separada entre ellos.
- Son multifuncionales.
- El nivel de escalabilidad en más fácil

Despliegue de Microservicios.

En (IBM Corp, 2015), se indica que para diseñar una arquitectura de microservicios es necesario incluir:

- Planificar un modelo de despliegue del microservicio.
- Determinar su organización y gestión el desplegar cada microservicio.
- Definir las herramientas tecnológicas que se van a usar.

Para los despliegues, estos se pueden realizar a través de 2 formas que son contenedores o máquinas virtuales.

Máquinas Virtuales

Cada máquina virtual se ejecuta con su propio sistema operativo (SO). Es responsabilidad brindar los recursos necesarios como el procesamiento y almacenamiento, incluido a las configuraciones de red que permitan estar expuestos al exterior.

Contenedores

Cada aplicación se ejecuta en un contenedor con su respectiva imagen (Ver ilustración 6). Solo se debe proveer la aplicación ya que el tiempo de ejecución y el servidor de la aplicación están incluido en cada contenedor, lo que facilita en gran manera el despliegue y el mantenimiento.

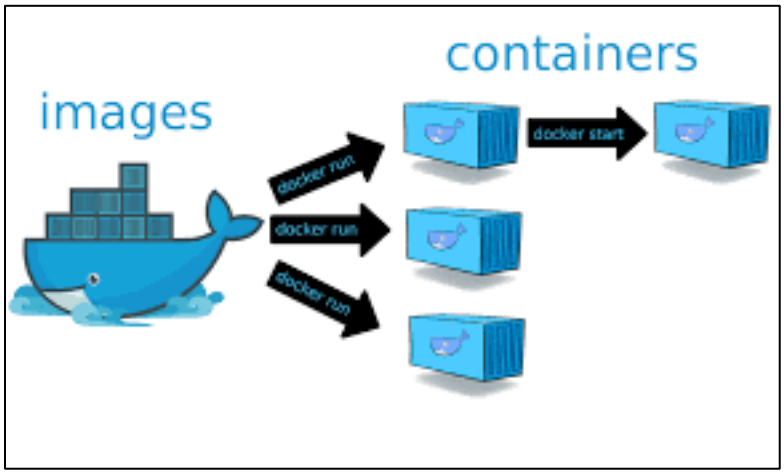
Un contenedor puede realizar las pruebas necesarias en entornos cerrados ya que, si llega a ocurrir un error o fallo, este se desplegará nuevamente.

Contenerizar es un procesos del desarrollo en la cual las aplicaciones, dependencias y configuraciones de entorno son empaquetados dentro de un conjunto (imagen del contenedor), probados independientemente y desplegados en el ambiente.

En este enfoque orientado a contenedores se puede eliminar la mayoría de los problemas que surgen al tener configuraciones de entorno incoherentes y los problemas que se derivan de ellos.

Figura 6.

Imagen de un contenedor



Nota: (Ampalio, 2020)

Los contenedores fragmentan la aplicación en un sistema operativo compartido. Con esto se estandariza el despliegue de la aplicación, accediendo que las aplicaciones se ejecuten como contenedores de Windows o Linux. Dado que el núcleo del sistema operativo (Linux o Windows) reparte a los contenedores su poder de procesamiento, ayudan a ser más ligeros que las imágenes de máquinas virtuales (VM).

Otra ventaja de utilizar contenedores a diferencia de una máquina virtual es la capacidad de instanciar rápidamente y con gran facilidad.

Sus principales beneficios son la agilidad, escalabilidad y portabilidad que brindan a la aplicación web.

Docker.

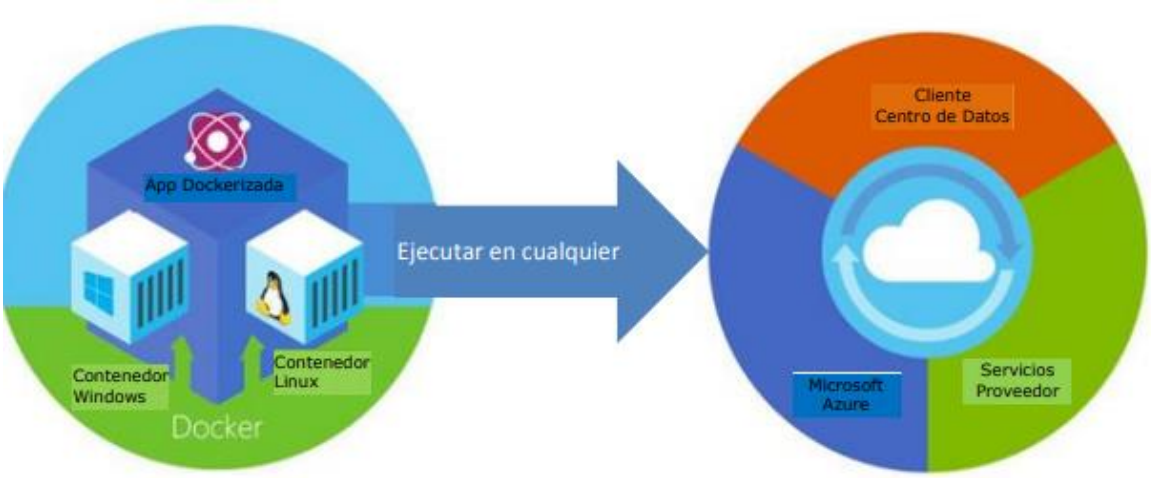
Docker es un proyecto “open source” que se creó para automatizar el despliegue de aplicaciones en contenedores portátiles y puedan ejecutarse en un proveedor de nube o localmente en una computadora.

Poco a poco viene a ser una buena estrategia la implementación de dockers ya que está siendo adoptada por la mayoría de los proveedores de plataformas: AWS, Microsoft Azure, etc.

La construcción y ejecución de contenedores se la puede realizar en cualquier sistema operativo, como se muestra en la ilustración 7.

Figura 7.

Despliegue en contenedores



Nota: Adaptado de (De la Torre, 2016)

En relación con los contenedores de Windows existen dos tipos:

- Contenedores de Windows Server
- Contenedores Hyper-V

Componentes

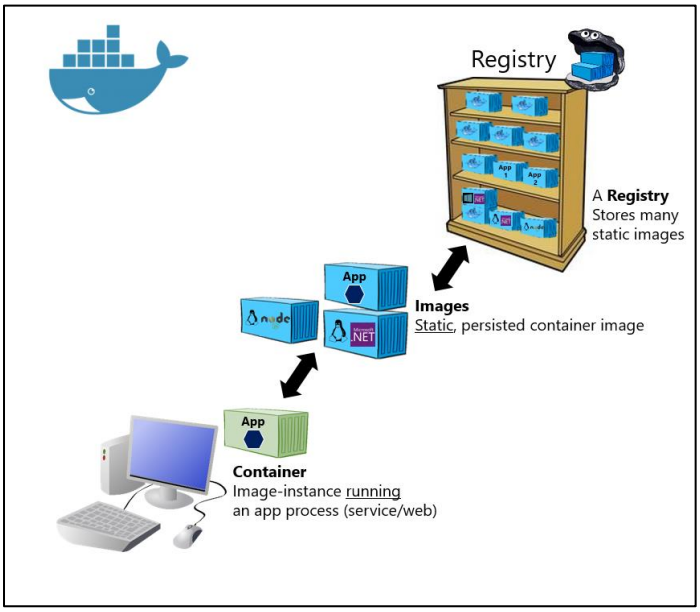
En la ilustración 8 se describen los componentes:

- **Contenedor**: Una o más instancias compuestas una imagen de Docker donde se almacena la aplicación o servicio.

- **Imagen**: Es una compilación dele aplicativo en el sistema de archivos y los parámetros proporcionados para el uso en tiempo de ejecución en un contenedor.
- **Repositorio**: Es un servicio que contiene repositorios de imágenes de docker y que pueden ser accedidos desde internet, hay 2 tipos de repositorios públicos y privados.

Figura 8.

Componentes de Docker



Nota: Adaptado de (De la Torre, 2016)

Diseño de comunicación entre servicios

Para la comunicación entre servicios se utiliza los mensajes por colas, ya que resultan ser suficientes, pero para las comunicaciones complejas se puede realizar por otros enfoques, los cuales son:

1. Los protocolos HTTP sincrónicos, como protocolo de acceso simple a objetos (SOAP), Transferencia de Estado Representacional (REST) o WebSockets, que ayudan a conservar el canal de comunicación entre el servidor y el cliente web (navegador) y manejar los eventos solicitud/respuesta (request/ response).

2. Mensajería asíncrona, a través de un broker o intermediario de mensajes

Para el tipo de arquitectura basada en microservicios generalmente se utiliza el estilo REST con referencia al protocolo HTTP.

Patrones de Arquitectura

El patrón arquitectónico que más se adecue a una estructura específica debe ser la actividad fundamental para las organizaciones o equipos de trabajo, en específico antes de empezar el diseño de las aplicaciones. En situaciones es necesario combinar más de un patrón, todo varía dependiendo del dominio del problema y sus requerimientos.

El patrón representa la relación entre un contexto, un problema y una solución, que ya fue solucionado con anterioridad, tal como se detalla en la Tabla 5.

Tabla 5.

Esquema de un Patrón

Contexto	Es el ambiente en el cual sucede el problema
Problema	Acción o conjuntos de acciones que genera conflictos a una o varias personas
Solución	Es un conjunto de actividades o tareas que erradica el problema. Ésta abarca: • Estructura con componentes y relaciones • Comportamiento a tiempo de ejecución: aspectos dinámicos de la solución

Fuente: propia

Dentro de los patrones arquitectónicos se pueden mencionar:

- Capas (Layers)
- Tuberías y Filtros (Pipes and Filters)
- Pizarra (Blackboard), Broker
- Modelo-Vista-Controlador (ModelView–Controller).

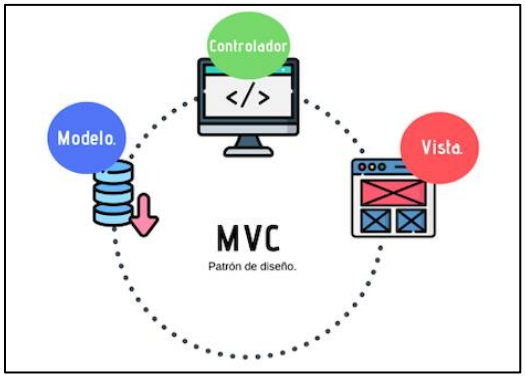
Patrón arquitectónico Modelo Vista Controlador.

El patrón Modelo Vista Controlador (MVC) está constituido por las siguientes estructuras:

- **Modelo**: Maneja los datos necesarios para sistema, es una representación de la base de datos.
- **Vista**: La interfaz e interacción con el usuario.
- **Controlador**: Es el intermediario entre el modelo y la vista. Gestiona la información y controla los datos entre ellos.

Figura 9.

Modelo vista controlador (MVC)



Nota: Adaptado de (García, 2019)

Diseño para una Estructura en Capas.

En la guía de estudio de la Arquitectura de Software de (Orjuela, Silva, & Castillo, 2019), hace referencia a que cada capa representa un conjunto lógico o un conjunto de servicio, que ayudan a diferenciar los tipos de actividades o tareas efectuadas por los módulos, y esto facilita a la implementación de un diseño que permite la reutilización de componentes. Es preciso destacar que estas capas no refieren con la ubicación física de los componentes.

Para el diseño son necesarios los siguientes pasos:

1. Seleccionar la táctica de estratificación respectiva.
2. Establecer las capas que se requiere.
3. Solventar la manera en que se hará la distribución de las capas.
4. Determinar si es preciso destruir capas.
5. Fijar reglas que sirvan para la intercomunicación entre capas.
6. Delimitar las interfaces entre capas.
7. Elegir adecuados protocolos de comunicación.

Capa de Presentación

Esta capa presenta o maneja todo lo relacionado a la interfaz de usuario (UI) y a la experiencia de usuario (UX), contiene los componentes que son necesarios para interactuar con el usuario final.

- **Elementos UI**: son los elementos que conforman el diseño.
- **Componente lógicos**: La lógica del negocio.

Capa de Negocio

En esta capa se establecen las reglas para realizar el diseño del aplicativo. La intención del arquitecto radica en reducir la complejidad al efectuar la división de las actividades o tareas en diferentes áreas de interés.

Capa de Datos

La capa de datos generalmente está integrada por:

a) Componentes de acceso a datos

b) Agentes de servicio. A continuación, se detallan los pasos a seguir:

1. Confeccione un diseño general que permita el acceso a datos.
2. Seleccionar los tipos de organismos que requiere.
3. Seleccionar el tipo de tecnología a usar para el acceso a los datos.
4. Plantee los componentes de acceso a datos.
5. Plantee los agentes de servicio.

Capa de Servicios

En la capa de servicios se efectúa y determina el medio de transmisión del servicio y las políticas o contratos de datos. Estos contienen:

- Interfaces de servicio.
- Tipos de mensajes.

La implementación se puede ejecutar de 2 formas distintas, que son: REST y SOAP; la primera opción permite a un conjunto restringido de operaciones, que se aplican a los recursos representados; en cambio la segunda opción consigue el desplazamiento por medio de diversos estados y la interrelación con un punto final.

Metodologías Ágiles

Según (Manifesto for Agile Software, 2018), los principios se enfocan en la colaboración y la comunicación, el funcionamiento del software, la flexibilidad y la organización del equipo para adecuarse a las realidades de los negocios.

Conocer las metodologías como Scaled Agile (SAFe), Disciplined Agile Delivery (DAD) y Scrum servirán de apoyo para implementar una Cultura de DevOps. El presente trabajo basará sus procesos en Scrum, pero se considera necesario presentar globalmente la definición de SAFe el cual incluye obligatoriamente a los DevOps en su framework.

Scrum

Según (McCarthy, R, 2020) lo define como una metodología que sirve para el desarrollo de software de manera ágil, es un conjunto de responsabilidades y roles que permiten el aprendizaje y descubrimiento continuos. La gestión del tiempo es una de las partes más importantes para Scrum y para llevar esto a cabo ha fijado un intervalo de tiempo para ejecutar los procesos y actividades dentro de los proyectos, garantizando así que el equipo no sobrepase los límites establecidos para determinados trabajos y evitando los desperdicios de tiempo y energía de los miembros del equipo.

Sprints

Scrum divide el tiempo en periodos cortos de trabajo conocidos como Sprints, generalmente en dos o cuatro semanas de duración.

Principios de Scrum.

En (Satpathy, 2018) se exhiben los seis principios que se aplica en Scrum:

1. Revisión del proceso empírico.
2. Autoorganización.
3. Colaboración.
4. Priorización basada en el valor.

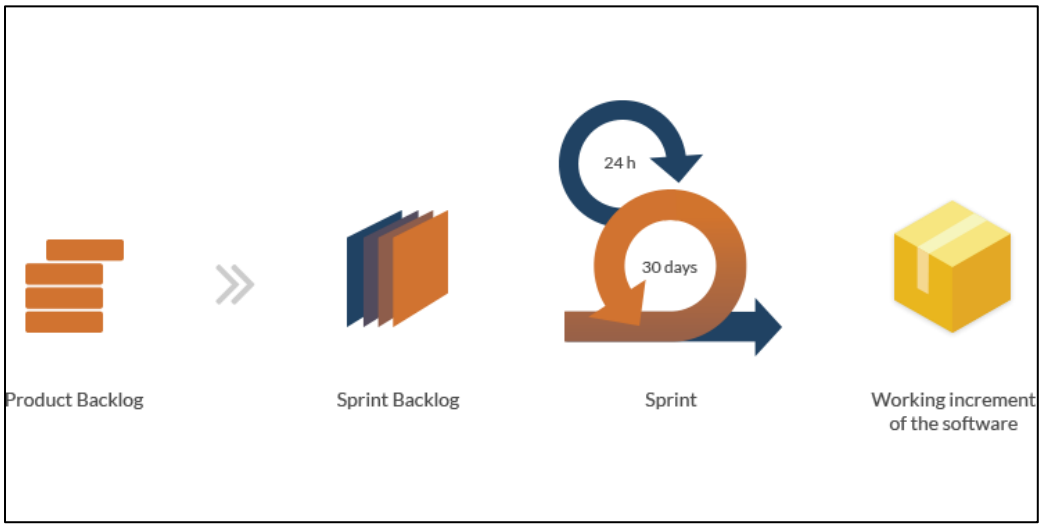
5. Asignación de un bloque de tiempo.
6. Desarrollo iterativo.

Artefactos Scrum

Los artefactos del marco de trabajo de Scrum serán aprovechados para el desarrollo de los capítulos posteriores ya que son la base del marco de trabajo de esta metodología.

Figura 10.

Artefactos de Scrum



Nota: Adaptado de (SoftwareCamp, 2015)

En la ilustración 10 se muestra componentes y flujo de trabajo de scrum, en donde inicia con el backlog del producto, la planificación del sprint, definición del sprint backlog, los Daily Scrum, el Incremento del producto, la revisión del sprint y la retrospectiva del sprint, sustentando de esta manera todo el ciclo de trabajo.

A continuación, se describe cada componente con las definiciones mencionadas por (Satpathy, 2018) en la guía para el cuerpo de conocimiento de Scrum presentados en la Tabla 6.

Tabla 6.

Componentes del Framework de Scrum

Componente	Descripción
Backlog de Producto	Es el listado donde se prioriza las funcionalidades del producto, y que ayuda al entendimiento de lo que se debe construir.
Planeación del Sprint	El Scrum Máster y el equipo de desarrollo son responsables de este componente, de tal manera que se planifica como se puede ejecutar los trabajos en cada Sprint y lo que se puede lograr.
Sprint Backlog	Se describe en un conjunto de tareas como diseñar, integrar, construir y probar el conjunto de ocupaciones de la cartera de productos de un sprint.
Daily Scrum	Es la actividad que se realiza todos los días con todos los miembros del equipo de desarrollo, que dura máximo 15 minutos

Incremento del Producto	El incremento del producto se refiere a los resultados del sprint.
Revisión del Sprint	Su objetivo es supervisar, monitorear y verificar el producto que se está construyendo, es necesario que intervengan los clientes, los patrocinadores y los miembros del equipo de Scrum.
Retrospectiva del Sprint	Su objetivo es inspeccionar el proceso durante el sprint. Todos los interesados incluido el equipo de desarrollo se reúnen en esta actividad.

Fuente: Adaptado de (Satpathy, 2018)

Cultura DevOps

DevOps es de los conceptos más mencionados en el entorno actual de TI, que por lo general está ligado a estrategias de transformación digital y a metodologías de desarrollo ágil.

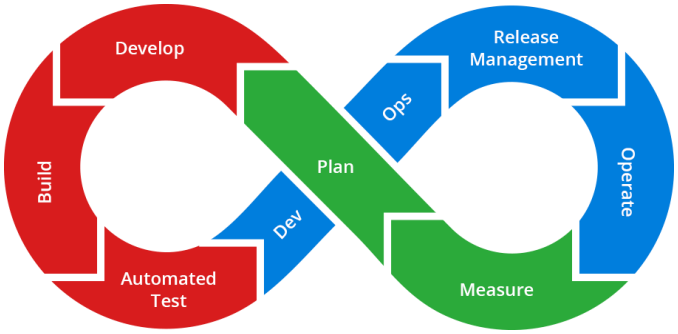
Según (Ravichandran, Taylor, & Waterhouse, 2016) definió: "*Los conceptos incluidos por DevOps, incorporan una reformulación en el contexto de la producción y el soporte de software. En lugar de mantener aplicaciones de ingeniería ("Dev") y gestión de TI ("Ops"), DevOps establece el uso de equipos más reducidos con experiencia multifuncional para perfeccionar la funcionalidad del software y los procesos utilizados a la hora de entregarlo*".

DevOps es un enfoque que tiene como objetivo la colaboración entre departamentos del negocio, desarrollo y operaciones de TI de manera eficiente y óptima. Es una estrategia empresarial que faculta la entrega continua, el despliegue continuo y la supervisión continua de aplicaciones. Reduce el tiempo necesario para tratar el feedback de los clientes. El desarrollo y

las operaciones, e incluso las pruebas, antes se organizaban en silos. DevOps las reúne para mejorar la agilidad.

Figura 11.

Visión general en DevOps



Nota: Adaptado de (Deloitte, 2019)

Principios DevOps

1. Acción enfocada al cliente. Abstraer y brindar medidas concentradas en el cliente a fin de que éste invierta en servicios y/o productos.
2. Responsabilidad de extremo a extremo. Brindar ayuda de la eficiencia y rendimiento durante todo el ciclo de vida del producto.
3. Mejora continua. Persistentemente se acelera la mejora de servicios y/o productos a fin de reducir perdidas.
4. Automatización global. No solamente en el desarrollo del software sino también con todo el panorama de la infraestructura.
5. Trabajo en equipo. En donde se establece cada rol, y se busca trabajar en total colaboración.
6. Supervisión y Monitoreo. Es significativo que exista un sólido procedimiento y monitoreo de pruebas

Ciclo de vida DevOps

1. Desarrollo. En esta etapa se construye el producto dando ciclos repetitivos
2. Prueba. En esta etapa, el equipo de QA se encarga de identificar, corregir y mitigar errores en cada nueva pieza de código y que debe ser realizado de manera rápida y eficaz, ayudado de herramientas de automatización.
3. Integración. En esta etapa, cada funcionalidad que ya ha sido probada se integra al código vigente y de nuevo se llevan a cabo las pruebas con la integración completa. El desarrollo continuo es posible gracias a la integración y pruebas continuas.
4. Despliegue. En esta etapa, se despliega a los distintos servidores, de manera que cualquier cambio o modificación realizada y en cualquier momento no afecte el código ni a las funcionalidades vigentes.
5. Monitoreo. En esta etapa, el equipo de operaciones se encargará del supervisar y monitorear el comportamiento de las aplicaciones desplegadas.

Proceso de desarrollo de software basado en DevOps

Para (Ravichandran, Taylor, & Waterhouse, 2016) el contexto aplicable a DevOps puede ser desconocidas para las empresas (en lo que se refiere a la rapidez de implementación, el porcentaje y tasa de cambio y la capacidad de respuesta hacia el cliente), por lo que es primordial analizar de manera general cómo los cambios en las prácticas de trabajo, el proceso y la tecnología pueden respaldar estos objetivos:

- **Personas**

Son indicadores de cambio potentes sin embargo son las complejas de recoliar. Se debe procurar mucho cuidado a las métricas internas, como las tasas de la capacitación y de retención de personal, junto con la formación de conocimientos y mentores.

Se abarca más que solo las capacidades del equipo de TI. Este paso establece una estrategia con la participación de la gerencia superior para que funcione, el principal propósito de todo esto es conseguir que todos sigan la misma orientación como:

1. El diseño de la organización.
2. Las estructuras departamentales.
3. Los roles y las responsabilidades laborales
4. Las líneas de generación de reportes.
5. La manera y forma en que se premia o incentiva a los empleados

A través de la cultura DevOps, se elimina la brecha que existe entre el ámbito del desarrollador y el personal de operaciones y ayuda a que las prácticas y ventajas de DevOps se difundan y sean practicadas por la organización en general.

El objetivo más importante para un equipo DevOps es que exista colaboración, comunicación e integración. Adicional a esto metodologías ágiles como scrum dan un complemento potente a la implementación efectiva de una cultura DevOps.

Según el Informe de Estado de DevOps en el 2017 realizado por (Puppet, 2017) se pudo observar que la demanda por personas conocedoras de DevOps está creciendo rápidamente, ya que las empresas que lo aplican obtienen excelentes resultados, logran implementar código con una frecuencia 30 veces mayor que sus competidores y sus fracasos de implementación son 50% menores.

Proceso

Los procesos establecen lo que la gente va a realizar. Una organización puede poseer una gran cultura de colaboración entre sus integrantes, sin embargo, si la gente realiza las cosas mal es un camino directo al fracaso.

Es relevante fijar que las prácticas existentes ayudarán o dificultarán el logro de nuevas metas u objetivos, haciendo énfasis en los cuellos de botella existentes como son seguridad, disponibilidad, etc.

Según (Menzel, 2015) en una cultura DevOps las actividades y los procesos estarán alineados junto con las obligaciones y el conocimiento en las personas correctas.

Es importante considerar la organización y sus procesos empresariales, desde la identificación y la priorización de requisitos hasta los lanzamientos de software y los procesos operativos, como la administración de cambios e incidentes.

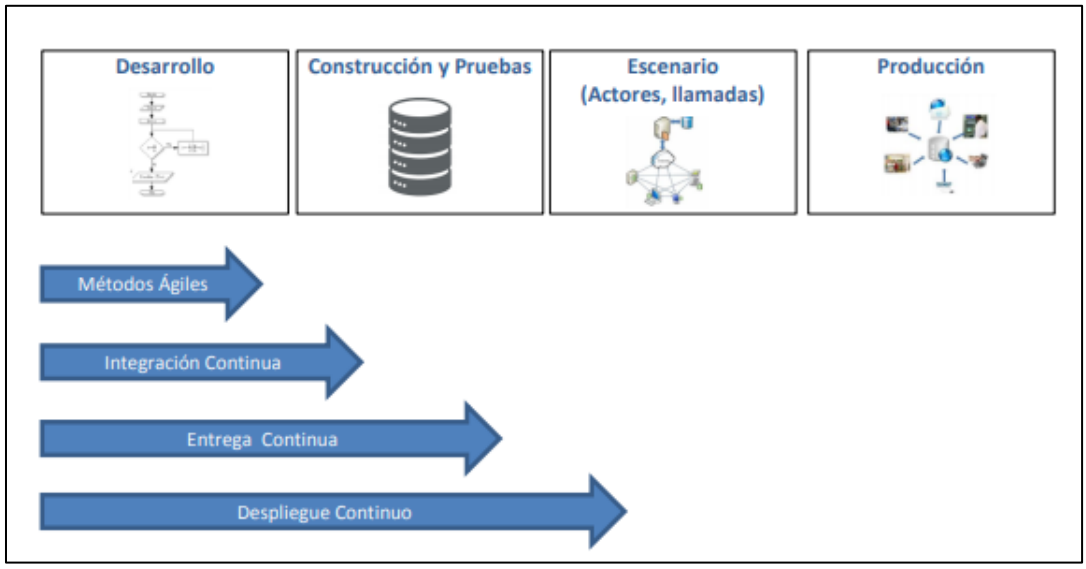
Para implementar DevOps, se deben considerar ciertos aspectos como:

- Métodos Ágiles.
- Integración Continua.
- Entrega Continua.
- Despliegue Continuo.

La ilustración 14 se muestra las etapas de las técnicas de DevOps, que muestra las herramientas de DevOps y los niveles de automatización que se pueden alcanzar usando procesos.

Figura 12.

Procesos en DevOps



Nota: Propia

Tecnología

En DevOps 2.0 Toolkit de (Farcic, 2017) expresa que para alcanzar una transformación digital exitosa utilizando DevOps, se debe analizar primero la infraestructura tecnológica que maneja la empresa y su capacidad que entrega, dando a consideración que las aplicaciones actuales que se encuentran en producción suelen ser monolíticas y se las debe desglosar o actualizar a soluciones que resuelvan el problema de raíz.

Las buenas métricas son aquellas que ayudan a los equipos a promover o estimular mejoras, incluso después de fallas, por ejemplo, cuál es el porcentaje de versiones fallidas y qué porcentaje de éstas se debió a defectos de código, errores de configuración, etc.

Las pruebas, la configuración y el despliegue pueden ser automatizadas a través de tareas, lo que libera al personal para centrarse en otras actividades valiosas y ayudando a reducir la posibilidad de error humano, según las recomendaciones dadas por NIST Cloud Computing

Reference Architecture por (Liu, y otros, 2011) la infraestructura en la nube (cloud) pueda perseguir los ritmos que demandan las aplicaciones en el mercado actual.

La infraestructura como código según (Swartout, 2014) es una capacidad básica de DevOps que permite a las organizaciones gestionar la escala y la velocidad, qué entornos necesitan ser aprovisionados y configurados para permitir la entrega continua.

Se dispone de tres tipos de herramientas de automatización para trabajar con infraestructura como código:

1. **Herramientas centradas en aplicaciones o middleware:**
2. **Herramientas de despliegue y medio ambiente:**
3. **Herramientas genéricas:**

Integración Continua

Se describe a que cada vez que alguien dele equipo de desarrollo realiza algún cambio o modificación en el código, toda la aplicación se vuelve a construir y se ejecuta el conjunto de pruebas. Y si algo falla en el proceso de construcción o pruebas, el equipo de desarrollo detendrá lo que se está haciendo para corregirlo inmediatamente.

La integración continua brinda las pruebas y organización de cada cambio conociendo el momento en que podría fallar para poder arreglarlo de inmediato

Requerimientos de la Integración Continua

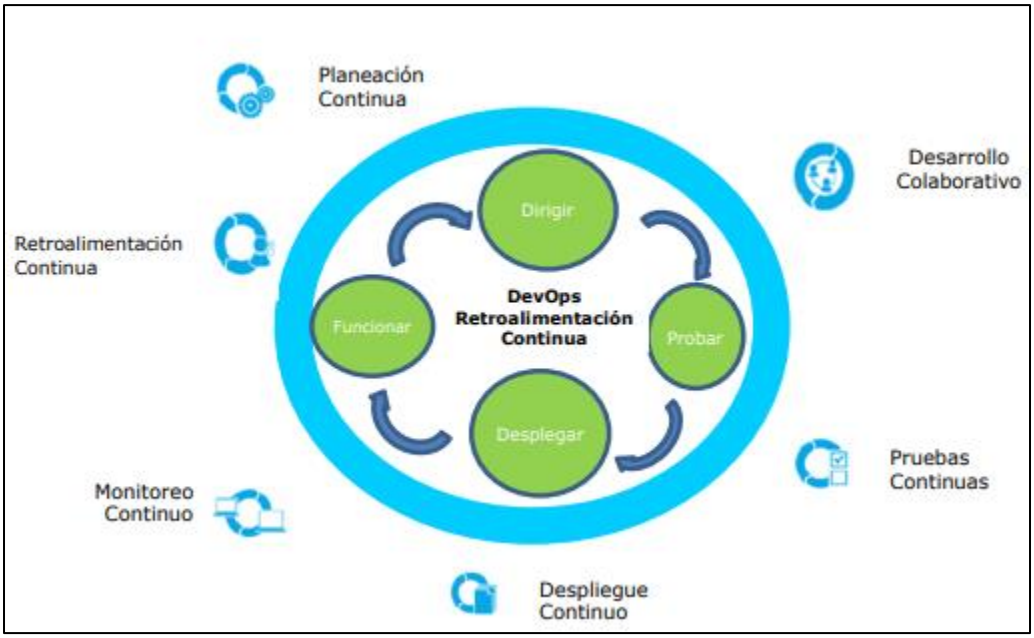
Estos son algunos requerimientos que se deben tener en cuenta para la Integración Continua:

1. Control de Versiones.
2. Una estructura automatizada.
3. Acuerdo del Equipo.

Finalmente revisados los conceptos de estas prácticas ágiles, es importante hacer énfasis que para implementarlas es necesario que el personal responsable tenga un rol DevOps y que tengan conocimientos sólidos en cuanto a la calidad en el desarrollo de los sistemas. Ver ilustración 13.

Figura 13.

Arquitectura de referencia de DevOps



Nota: Adaptado de (Sharma & Coyne, 2015)

Entrega Continua

Es la práctica que bajo la cultura DevOps, el cual ha sido pensada y diseñada para brindar un mayor rendimiento y eficiencia, y que el equipo de desarrollo y operaciones se mantengan en una retroalimentación y capacitación constante sobre la aplicación. Las aplicaciones deben estar siempre en un estado disponible para el despliegue.

Tanto la empresa como el equipo de TI pueden probar la aplicación en cualquier momento. Se tiene un sistema de registro de las versiones de cada aplicación, mediante el uso

de herramientas adecuadas que permiten llevar un control desde el primer momento de la entrega.

Según (Fowler, 2014) "*la entrega continua es la práctica de desarrollo de software en la que la aplicación es construida de una forma que puede ser liberada a producción en cualquier momento*".

Principios de la Entrega Continua

En (Humble & Farley, 2016) se mencionan ocho principios, los cuales son:

1. Crear un proceso repetible y confiable para la liberación del software.
2. Automatizar todo lo que se pueda.
3. Mantener todo dentro de un control de versiones.
4. Si duele, hágalo con más frecuencia y traiga el dolor hacia adelante.
5. Construir con calidad.
6. Hecho significa liberado.
7. Todo el mundo es responsable del proceso de entrega.
8. Mejora Continua.

Despliegue Continuo

En esta práctica se establece y supervisa el pipeline trazado en un inicio y se lo aplica en producción. De esta manera si el monitoreo pasa las pruebas requeridas se procede con el despliegue en el ambiente requerido directamente.

Los proveedores de nube existentes son: Amazon Web Services, Azure y Google.

Tabla 7.

Servicios de proveedores de nube

Servicio	Amazon Web Services	Azure	Google Cloud Platform
IaaS – Despliegue y gestión de servidores virtuales	Elastic Compute Cloud (EC2)	Azure Virtual Machines	Compute Engine
Servidores virtuales privados	Lightsail	Virtual Machine Images	
Servicio para contenedores	EC2 Container Service (ECS)		
Servicio para Kubernetes	Elastic Container Service para Kubernetes (EKS)	Azure Kubernetes Service (AKS)	Kubernetes Engine
Registro de contenedores Docker	EC2 Container Registry (ECR)	Azure Container Registry	Container Registry
Servicio para orquestar aplicaciones basadas en microservicios	Service Fabric	App Engine	

Escalado automático	AWS Auto Scaling	Virtual Machine Scale Sets	
Servicio de backup	Site Recovery		
Autenticación y autorización	Identity and Access Management (IAM)	Active Directory	
Cifrado	AWS Key Management Service	Key Vault	
Protección frente a ataques por denegación de servicio distribuido (DDoS)	AWS Shield	DDoS Protection Service	Cloud Armor
Precios	S**e** ofrecen niveles de servicio gratuitos para personas particulares o empresas nuevas. Donde se puede contratar servicios por segundo, en lugar de por hora.	Dependerá de los servicios que necesite el equipo. El costo del servidor puede variar desde $0.099 por hora a $0.149 por hora	La plataforma tiene precios de pago a medida que se avanza y facturación “por segundo” de uso. Además, ofrece descuentos para el uso a largo plazo que comienza después del primer mes

Fuente: Adaptado de (DWS Serban Group, 2020)

En base al cuadro comparativo se tomó la decisión de elegir los servicios de Amazon web services, por la cantidad de servicios que se necesita, y su modelo de precios “por segundo”, el cual se ajusta a las necesidades del proyecto.

Pipeline

El pipeline es el conjunto de instrucciones o tareas del proceso que sigue una aplicación, a partir del repositorio de control de versiones hasta que llega a los usuarios. Cada actualización en el software conlleva un complejo proceso hasta ser desplegado. (Moreno, 2019)

El proceso incluye a partir de la construcción del software de forma fiable, hasta realizar todas las pruebas y despliegues necesarios. Un pipeline proporciona un conjunto de herramientas que da forma la hora de entregar la aplicación.

Un Pipeline aumenta un conjunto de herramientas de automatización que soporta varias cargas de trabajo, desde trabajos simples hasta complejos pipelines de entrega continua.

Como resultado da una serie de tareas relacionadas, los usuarios pueden aprovechar muchas de las características de un pipeline como:

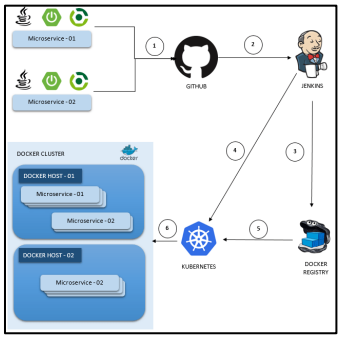
- Código: los Pipelines Jenkins se implementan en el código y suelen incorporarse al proyecto a través del sistema de control de versiones
- Durable: Los pipelines, perduran tanto a los reinicios planificados como a los no planificados del servidor

- Pausable: Opcionalmente, pueden parar y esperar la intervención o la aprobación de una persona, antes de continuar su ciclo.
- Versátil: Soportan complejos requisitos de despliegue continuo, incluyendo la habilidad de hacer forks, loops, o realizar trabajo en paralelo.
- Extensible: Además, también soporte extensiones personalizadas de su lenguaje específico de dominio, así como múltiples opciones para la integración con otros plugins.

En la ilustración 14 se puedes ver un ejemplo de escenario de despliegue continuo configurable en un Pipeline Jenkins.

Figura 14.

Despliegue continuo usando pipelines



Nota: (Betsol, 2020)

En el paso 1 se sube los diferentes microservicios al respectivo repositorio de código en este GitHub, a lo cual en el paso 2, Jenkins usando pipelines empieza a ejecutar las diferentes tareas, en el paso 3, se registra un docker para Contenerizar cada microservicio, en el paso 4 y

5 se inicia con la orquestación de los contenedores a través de kubernetes, usando un docker clúster.

Herramientas

Herramienta para versionamiento de código

Es necesario el repositorio de control de versionamiento, el cual se utilizará GitHub, que actualmente se maneja en la empresa, adicional a eso ya posee licencias empresariales el cual nos brinda más servicios, entre los más importantes es GITHUB ACTIONS, el cual es el servicio que complementa integración y entrega continua.

Figura 15.

GitHub

Nota: (SoftwareCamp, 2015)

Herramienta para integración continua

Es necesario el software para el servidor de integración continua, donde se indicará las tareas para hacer la compilación automática, integrando con el repositorio del código, además de lanzar las pruebas unitarias si así se desea. En el mercado existen varias herramientas con este fin, los cuales son Jenkins, Travis CI y Team City entre las más potentes, sin embargo, Jenkins es gratuito, open-source y actualmente uno de los más empleados para esta función, por tal razón se utilizará Jenkins.

Figura 16.

Jenkins

Nota: (SoftwareCamp, 2015)

Herramientas para pruebas unitarias

Spring Test

Framework para realización de pruebas unitarias de aplicaciones hechas en base al framework spring

JUnit

JUnit es un conjunto de bibliotecas que son utilizadas en programación para la realización de pruebas unitarias en aplicaciones basadas en Java y que pueden ser utilizadas en sus diferentes frameworks.

Mockito

Es un marco de prueba de código abierto para aplicaciones basadas en Java. Este permite la creación de objetos de prueba (objetos simulados) en pruebas unitarias automatizadas con el objetivo del desarrollo promovido por comportamiento (BDD) o desarrollo promovido por pruebas (TDD).

Herramientas para escaneo de código

Al implementar las buenas prácticas de DevOps también es necesario evaluar y medir la calidad del código fuente, que se realiza a través de un análisis estático sobre dicho código, con

la meta de advertir sobre diferentes puntos a mejorar y obtener métricas que ayudan a mejorar el código. Para eso Sonarqube cuenta con análisis de código y su costo es relativamente bajo.

Figura 17.

Sonarqube

Nota: (SoftwareCamp, 2015)

Herramienta para notificaciones

Adicional a las herramientas es necesario de una herramienta para la colaboración entre los empleados de la empresa, también servirá para la notificación de problemas en tiempo real, en este caso la empresa ya cuenta con Slack, este servirá para la comunicación con todos los miembros.

Figura 18.

Slack

Nota: (SoftwareCamp, 2015)

Herramienta para monitorización

Grafana es la herramienta para representar los datos de serie temporales. A partir de una serie de datos recolectados se conseguirá un panorama gráfico de la situación actual de una empresa u organización.

Según lo propuesto por (Balestrini, 2016), el marco metodológico permitirá establecer el procedimiento teórico y técnico a seguir sobre la base del enfoque cualitativo, que permita proponer una arquitectura para la integración y entrega continua de aplicaciones web.

Operacionalización de variables

Las siguientes dimensiones e indicadores proporcionan las métricas para la evaluación de los procesos y que se detallan en la tabla 8.

Tabla 8.

Indicadores y métricas de variables

Dimensión	Indicadores	Unidad de Medida	Fórmula
Velocidad	Ciclo de vida: evalúa el tiempo desde el inicio del desarrollo hasta su despliegue en producción.	Días	CT = DD+DT+DP CT= Ciclo de vida DD= Días de Desarrollo DT= Días de Testing DP= Días de despliegue a producción
	Frecuencia de liberación de Código: analiza y evalúa la cantidad de veces que se entrega las historias de usuario por sprint	Número	CD=Cantidad de Despliegues por sprint

Calidad	Ratio de Éxito: Analiza y Evalúa el Porcentaje porcentaje de código que ha sido desplegado de manera exitosa en cada sprint.	RE = DE/TD*100% TD = DE+DF DE=Despliegue exitosos por sprint DF=Despliegues fallidos por sprint TD=Total de despliegues por sprint RE= Ratio de éxito

Fuente: propia

Capítulo III

Metodología

En este capítulo se detalla las fases que han sido ejecutadas en los Sprints para el desarrollo de la arquitectura propuesta.

Para ello, se aplicó un diseño preexperimental en la cual se recolectaron datos de la arquitectura actual y de la arquitectura propuesta para realizar el respectivo análisis comparativo, donde se obtiene resultados para dar validez o no de la hipótesis planteada, siendo el presente trabajo de tipo aplicado, según (Hernández Sampieri, Fernández Collado, & Baptista Lucio, 2014). Esto permite la utilización de teorías existentes para resolver el problema planteado.

El enfoque que se utilizó es cuantitativo ya que se adapta a las necesidades y características del caso práctico.

Metodología scrum

La metodología SCRUM tiene estas fases en las cuales se detalla el trabajo realizado estas son:

- **Planificación**

  En esta sección se detalla el objetivo de cada sprint también contiene los procesos o estrategias que se utilizaran para alcanzar los objetivos

- **Implementación**

  Aquí se implementa lo planificado en cada Sprint.

- **Retrospectiva**

  Esta sección contiene los diferentes inconvenientes y aciertos que ayudan a mejorar el desempeño del equipo con la finalidad de lograr los objetivos propuestos

- **Lanzamiento**

Este apartado contiene los entregables detallados en cada Sprint, estos entregables contemplan los criterios de aceptación necesarios para que cumpla con lo especificado en cada Sprint

Roles

Los roles manejados en la implementación de esta arquitectura se definen a continuación:

- Stakeholder, el cual es el interesado del producto.
- Scrum Máster, el cual tiene como finalidad que todo el equipo no tenga inconvenientes y se pueda cumplir con la implementación de una manera óptima.
- Product Owner, este rol se encarga de comunicar, representar al cliente para ello debe garantizar que se trabaje de manera adecuada.

Requerimientos

El primer levantamiento de requerimientos se realizó mediante entrevistas al área de sistemas de la empresa Emagic S.A y la participación de todo el equipo Scrum.

A continuación, en la tabla 9 y se detallan los requerimientos necesarios para el desarrollo y la implementación de la arquitectura.

Tabla 9.

Requisitos

ID	Requerimiento	Nivel
1	Aplicar buenas prácticas DevOps	Medio
2	Implementar contenedores	Alto
3	Implementar el versionamiento de código	Alto

4	Implementar la integración continua automática	Alto
5	Implementar notificación y escaneo de código automático	Medio
6	Realizar el despliegue de la aplicación mediante integración continua	Alto

Product Backlog

En la tabla 10 se detalla las historias de usuario necesarias para el desarrollo de la arquitectura propuesta.

Tabla 10.

Product Backlog

Id de la épica	Id de la historia de usuario	Enunciado	Estado	Esfuerzo	Prioridad
TCT-4	TCT-13	Revisión de la literatura.	Finalizada	4	Media
TCT-4	TCT-14	Revisión de metodología investigativa.	Finalizada	5	Media
TCT-4	TCT-15	Revisión de problemática actual de la empresa y arquitectura.	Finalizada	6	Media
TCT-4	TCT-16	Investigación de herramientas para CI/CD y análisis comparativo entre las mismas.	Finalizada	7	Media
TCT-5	TCT-17	Análisis financiero del proyecto	Finalizada	4	Alta

TCT-5	TCT-18	Análisis de lógica del negocio de microservicio y el cliente.	Finalizada	5	Media
TCT-5	TCT-19	Creación del microservicio en Spring Boot y cliente en Angular.	Finalizada	6	Alta
TCT-5	TCT-20	Codificación de pruebas unitarias utilizando Mockito Junit.	Finalizada	7	Alta
TCT-6	TCT-21	Creación de Docker.	Finalizada	4	Media
TCT-6	TCT-22	Creación y configuración de Jenkins	Finalizada	5	Media
TCT-6	TCT-23	Creación pipeline Jenkins basado en webhooks.	Finalizada	7	Media
TCT-6	TCT-24	Integración con pruebas automáticas	Finalizada	8	Alta
TCT-7	TCT-25	Revisión de la literatura.	Finalizada	6	Media

TCT-7	TCT-26	Revisión de la literatura.	Finalizada	5	Media
TCT-7	TCT-27	Integración con Jenkins y pipelines	Finalizada	3	Alta
TCT-7	TCT-28	Instalar Sonarqube e integrarlo con Jenkins	Finalizada	6	Media
TCT-7	TCT-29	Agregar escaneo de código con sonarqube	Finalizada	6	Media

Fuente: propia

Preparación del ambiente de desarrollo

En esta sección se detalla la preparación de los ambientes y componentes necesarios para implementar la arquitectura propuesta.

Creación de instancia AWS e instalación de contenedores.

La creación de Instancia en AWS servirá como servidor de integración continua donde se instaló los contenedores y además contempla las siguientes especificaciones para un mejor rendimiento al momento de desplegar las aplicaciones.

Tabla 11.

Tabla de características del servidor de Integración

Características de Aws	**Descripción**
Tipo de instancia	t2. xlarge
Tamaño del volumen (GiB)	8

Sistema operativo	Ubuntu

Fuente: propia

Adicional es necesario configurar el grupo de seguridad el cual permitirá la comunicación entre los aplicativos que ayudan a la integración y entrega continua.

Como se mencionó anteriormente el objetivo es tener una arquitectura basada en contenedores para lo cual es necesario los siguientes comandos necesarios para comprender el funcionamiento de los contenedores:

Tabla 12.

Comandos para iniciar y verificar estado de los contenedores

Comando	**Descripción**
docker build -t jenkins/77lueocean_ci –no-cache.	Crea una imagen basada en Jenkins además de Maven que es necesario para esto es necesario un Dockerfile.
Docker run -p 8080:8080 -p 50000:50000 –name jenkinsblue jenkins/blueocean_ci.	Permite *desplegar el contenedor, indica los puertos el nombre de la imagen y el nombre del contenedor*
docker ps -a.	Permite visualizar los contenedores que se utilizará para la integración y entrega continua.
Docker start -a <id del contenedor>.	Este comando es para inicializar el contenedor.

Docker exec -it jenkinsblue /bin/ash	Accede a la consola del contenedor.
Docker network create jenkins_sonarqube	Crea una red de contenedores para la comunicación.
Docker network connect jenkins_sonarqube sonarqube	Conecta el contenedor sonarqube a la red de contenedores
docker container inspect sonarqube	Obtiene los detalles del contenedor en formato JSON

Fuente: propia

Para obtener una arquitectura basada en contenedores son necesarios los siguientes contenedores detallados en la tabla 13.

Tabla 13.

Contenedores

Contenedor	**Descripción**
Contenedor de Jenkins	docker build -t jenkins/bueocean_ci --no-cache. (Creado mediante Dockerfile)
Contenedor de SonarQube	docker pull sonarqube docker run -d --name sonarqube -p 9000:9000 -p 9092:9092 sonarqube

Contenedor Minikube	docker pull Minikube

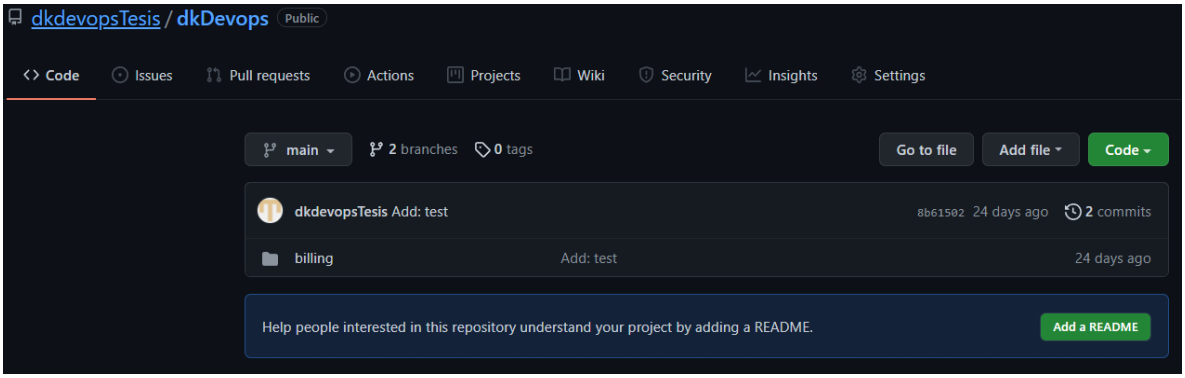
Fuente: propia

Creación de repositorio de código.

El repositorio de código es necesario para interactuar con los desarrolladores y el servidor de integración continua existen varios repositorios de código el utilizado en esta arquitectura es GitHub el cual alojara el código de los 2 proyectos uno desarrollado en SpringBoot y el cliente en Angular ya desarrollados.

Figura 19.

Repositorio de código GitHub



Nota: Propia

Planificación del lanzamiento

Con el fin de tener una visión global del desarrollo del proyecto se planifico que historias de usuario se van a desarrollar en cada Sprint tomando en cuenta los puntos de esfuerzo estos puntos de detallan en la tabla 14.

Cada Sprint tiene una duración de 3 semanas.

Tabla 14.

Puntos de esfuerzo total

Sprint	Historias de usuario	Puntos de esfuerzo total
1	TCT-10, TCT-11, TCT-12, TCT-13 TCT-14	23
2	TCT-16 TCT-17, TCT-18, TCT-19,	25
3	TCT-20, TCT-21, TCT-22, TCT-23, TCT-24, TCT-25, TCT-26, TCT-27, TCT-28, TCT-29	62
4	TCT-30, TCT-31	14
	Total, puntos de esfuerzo	**124**

Fuente: propia

Sprint 1

En este Sprint se realiza un análisis de la arquitectura actual de la empresa, se analiza las diferentes herramientas necesarias para implementar las buenas prácticas DevOps. Además, se organiza la arquitectura que será implementada.

3.6.1 Planificación del sprint 1

Objetivos del sprint 1

- Analizar arquitectura
- Analizar Herramientas
- Revisar metodología y literatura

Lista de historias y tareas del sprint 1

En la ilustración 20 se detalla las historias y las tareas a ejecutarse en el Sprint 1

Figura 20.

Tablero del Sprint 1

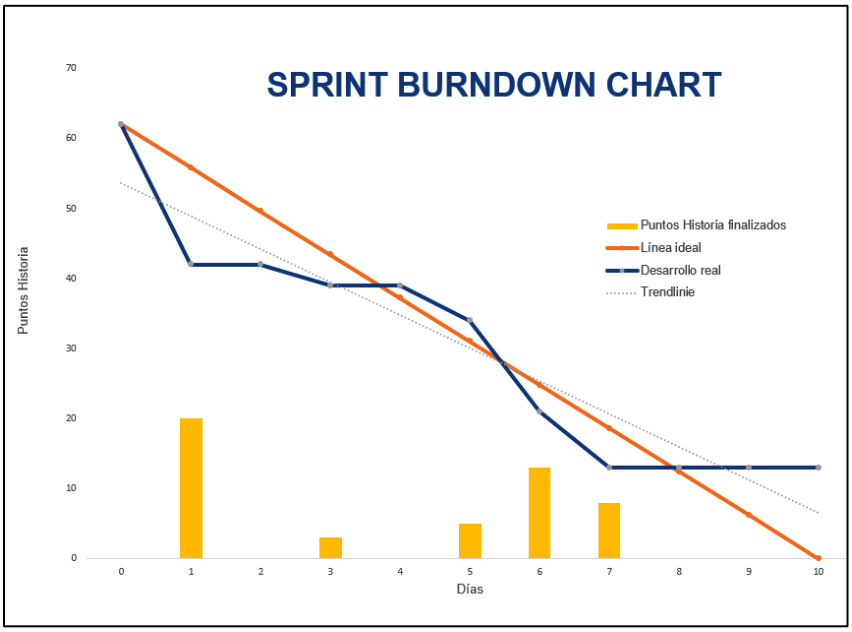
Nota: propia

Ejecución del sprint 1

A continuación, en la ilustración 21 se detalla cómo se ejecutó el sprint con las historias ya finalizadas.

Figura 21.

Burndown chart del Sprint 1



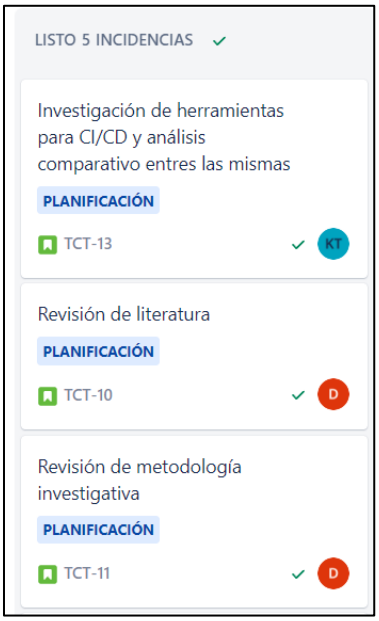
Nota*: propia*

Revisión del sprint 1

En este Sprint fue revisado por el equipo Scrum verificando el cumplimiento de cada historia de usuario mediante los criterios de aceptación planteados.

Figura 22.

Historias de usuario finalizadas Sprint 1



Retrospectiva Sprint 1

El sprint no tuvo ninguna incidencia lo cual se refino el producto Backlog. Con la finalidad que mejore la ejecución del siguiente Sprint

Sprint 2

En este Sprint se realiza la codificación de los aplicativos tanto el Front-End como el Back-End todo esto utilizando los Frameworks que hacen que el desarrollo sea mucho más ágil. Además, se codifica las pruebas unitarias que hacen parte de la Integración continua

Planificación del sprint 2

Objetivos del Sprint 2

- Construir el microservicio y el cliente
- Codificar las pruebas unitarias

Lista de historias y tareas del Sprint 2

En la ilustración 23 se detalla las historias y las tareas a ejecutarse en el Sprint 2

Figura 23.

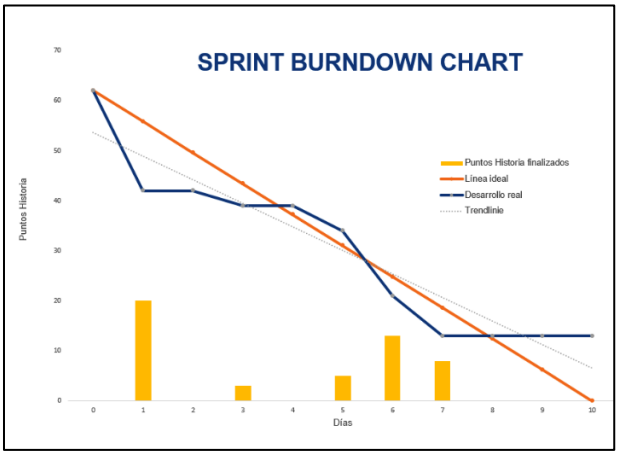
Tablero Sprint 2

Ejecución del sprint 2

Gráfico del burndown chart en cual muestra el progreso obtenido en el Sprint 2

Figura 24.

Burndown chart sprint 2



Revisión del sprint 2

En este Sprint el equipo Scrum verifico el cumplimiento de cada historia de usuario mediante los criterios de aceptación planteados.

Figura 25.

Historias de usuario finalizadas Sprint 2

Retrospectiva sprint 2

El sprint no tuvo ninguna incidencia lo cual se refino el producto Backlog. Con la finalidad que mejore la ejecución del siguiente Sprint

Sprint 3

En este Sprint se implementa la integración con Jenkins basado en webhooks se codifica los pipelines, se integra sonarqube para el escaneo de código y se implementa la integración con Slack.

Planificación del Sprint 3

Objetivos del Sprint 3

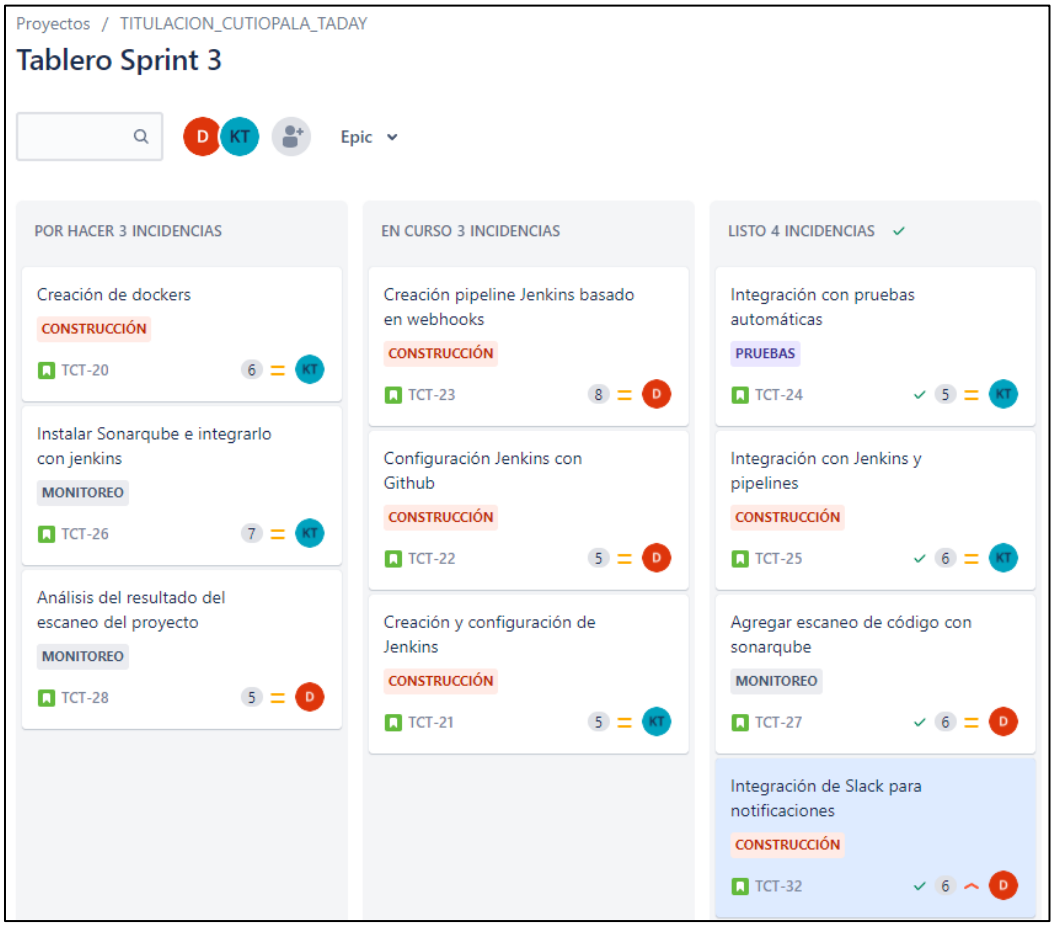
- Implementar la Integración continúa contemplando las buenas prácticas DevOps
- Implementar herramientas de código para el monitoreo

Lista de historias y tareas del Sprint 3

En la ilustración 26 se detalla las historias y las tareas a ejecutarse en el Sprint 3

Figura 26.

Tablero Sprint 3

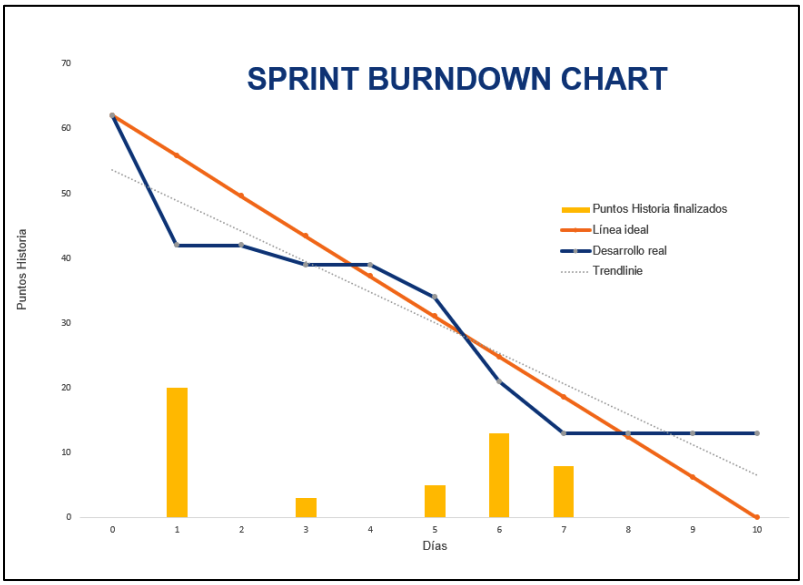


Ejecución del sprint 3

Gráfico del Burndown chart el cual muestra el progreso del Sprint 3

Figura 27.

Burndown chart Sprint 3

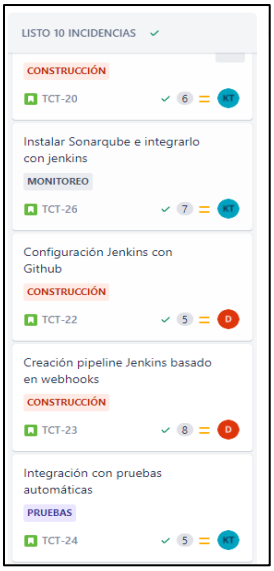


Revisión del sprint 3

En este Sprint fue revisado por el equipo Scrum verificando el cumplimiento de cada historia de usuario mediante los criterios de aceptación planteados.

Figura 28.

Historias de usuario finalizadas Sprint 3



Retrospectiva Sprint 3

El Sprint 3 se cumplió con los objetivos establecido, el único inconveniente que se encontró fue el planteamiento del grupo de seguridad en el servidor AWS, para que se comuniquen las diferentes herramientas de integración continua.

Sprint 4

En este Sprint se realiza el despliegue de la aplicación el cual se realizará en un clúster de Kubernetes. Para ello se creará un proyecto independiente el cual contendrá todas instrucciones necesarias para su funcionamiento.

Planificación del sprint 4

Objetivos del Sprint 4

- Implementar el despliegue automático
- Implementar herramientas de monitorización

Lista de historias y tareas del Sprint 4

En la ilustración 29 se detalla las historias y las tareas a ejecutarse en el Sprint 4

Figura 29.

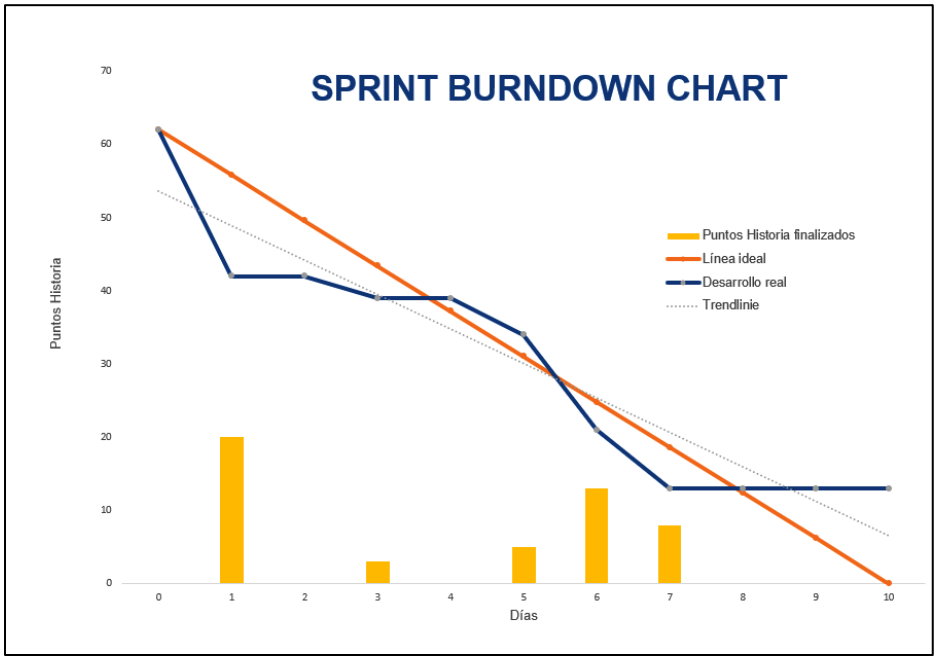
Tablero Sprint 4

Ejecución del sprint 4

Gráfico del Burndown chart el cual muestra el progreso del Sprint 4

Figura 30.

Burndown chart Sprint 4



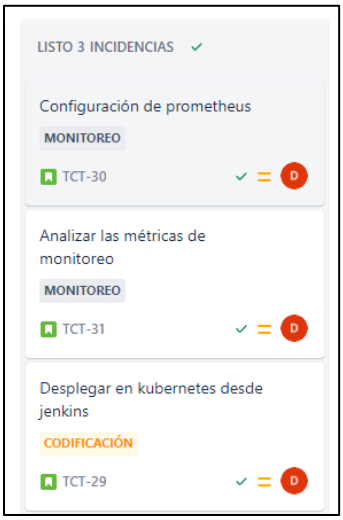
Nota: propia

Revisión del sprint 4

En este Sprint fue revisado por el equipo Scrum verifico el cumplimiento de cada historia de usuario mediante los criterios de aceptación planteados.

Figura 31.

Historias de usuario finalizadas Sprint 4



Nota: propia

Retrospectiva sprint 4

El sprint 4 se puede mejorar los scripts planteados para el despliegue de la aplicación e importante a mencionar es que la implementación del monitoreo se realizó de manera efectiva.

Capítulo IV

Evaluación análisis y resultados

En este capítulo se presenta los resultados de las pruebas de usabilidad y del monitoreo, para las pruebas de usabilidad se diseñaron casos de prueba los cuales se detallan en el anexo 3. Por otra parte, para analizar el monitoreo de la aplicación se tomó en cuenta las dimensiones e Indicadores necesarios para medir el rendimiento de la arquitectura antigua vs la implementada.

Pruebas de usabilidad

Las pruebas de usabilidad se aplicaron a 11 personas de la empresa Emagic S.A, entre los encuestados se encuentran 1 Arquitectos de Software, 5 desarrolladores Senior, 3 Analistas de Software, 3 clientes.

Con la finalidad de determinar el grado de satisfacción de los involucrados al implementar la arquitectura propuesta se planteó las siguientes preguntas evaluadas de forma cualitativa detalladas en la tabla 15.

Tabla 15.

Preguntas utilizadas para el área de sistemas.

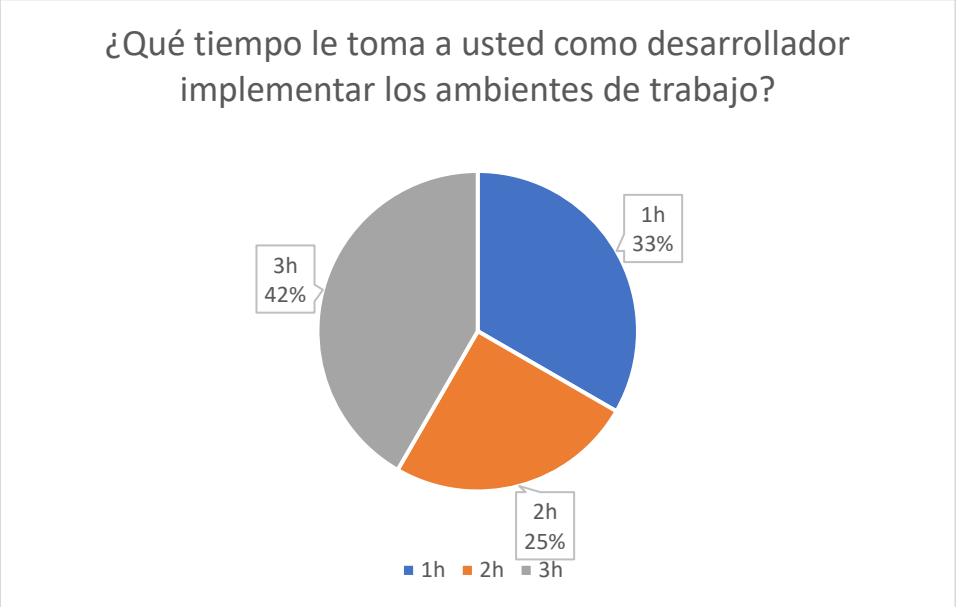
Pregunta	Descripción
1	¿Qué tiempo le toma a usted como desarrollador implementar los ambientes de trabajo?
2	¿Qué tiempo le toma a usted como desarrollador desplegar una aplicación?
3	¿Qué tiempo ocupa usted en capacitar al nuevo personal, en temas de despliegue de aplicaciones?

4	¿Qué tiempo ocupa usted al realizar un cambio en el proyecto y subirlo al ambiente de pruebas, certificación y producción?
5	¿Usted considera necesario el escaneo de código, el monitoreo de log y las notificaciones al momento de desplegar una aplicación?
6	¿Considera usted que la arquitectura propuesta mejora los tiempos de despliegue de las aplicaciones?
7	¿Considera usted que la arquitectura propuesta permite optimizar la calidad de código?
8	¿Considera usted que la arquitectura propuesta reduce errores de codificación y despliegue?

A continuación, se muestra los resultados obtenidos de las preguntas planteadas,

Pregunta 1

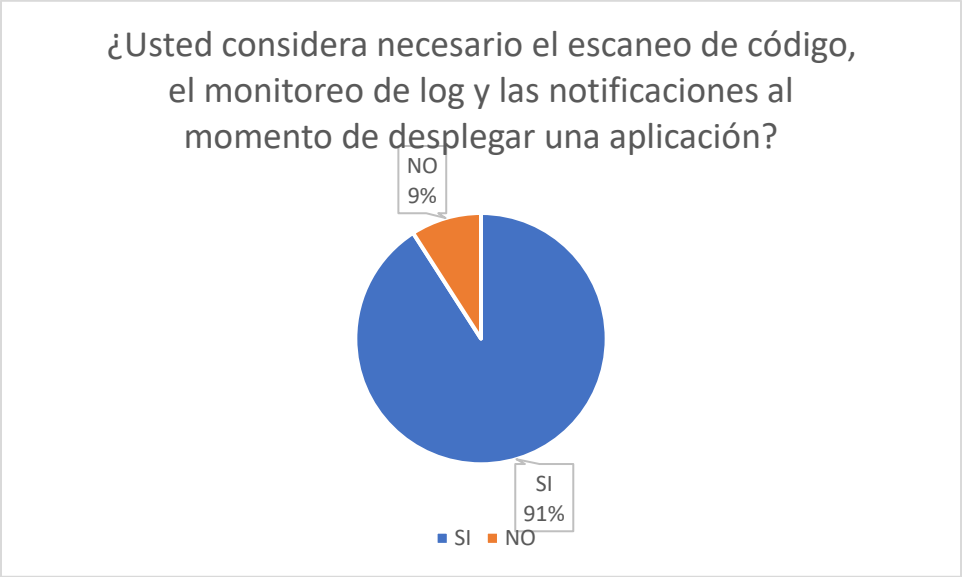
Figura 32. *Resultados pregunta 1*



En esta ilustracion se muestra que el 42% de los encuestados tienen problemas al preparar el entorno de trabajo ademas de que el tiempo es prolongado cerca a las 3 horas.

Pregunta 6

Figura 33. *Resultados pregunta 6*



Nota: Para visualizar el formulario ver el anexo 3

En esta ilustracion se muestra que un 9% no está de acuerdo ya que en observaciones se encontró el siguiente detalle: requiere más conocimiento implementar todas las herramientas para la integracion y entrega continua.

Resultados de monitoreo

Después de implementar la arquitectura propuesta con los contenedores y herraminetas definidas en el capítulo 2 se obtuvieron los siguientes resultados, en el período comprendido entre el 05/08/2021 al 05/09/2021.

Se evidencia un funcionamiento correcto sin mayores complicaciones. Para mayor detalle ver anexo 4.

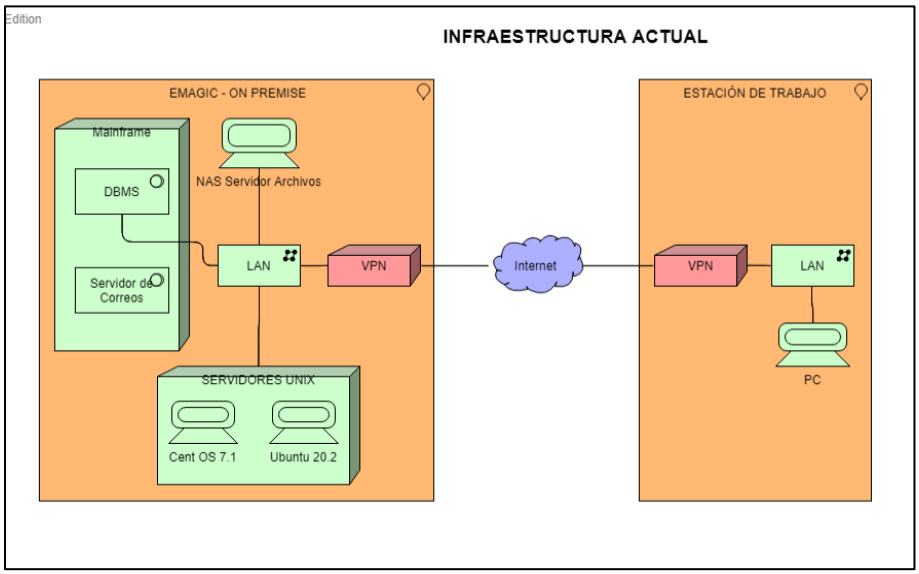
Resultados del proceso de implementación

Arquitectura actual.

La presente arquitectura muestra los diferentes componentes manejados por la empresa Emagic S.A la cual consta de un servidor Centos OS 7.1 en la cual se aloja el servidor de aplicaciones y la base de datos, la conexión con los clientes se realiza por medio de una VPN, actualmente el despliegue de una aplicación se la realiza de la siguiente manera:

- Se prepara el entorno, en la cual se instalan los diferentes aplicativos y dependencias necesarias.
- Ejecución de configuraciones y pruebas
- Corrección de problemas
- Cambio de estado en producción

Figura 34. *Infraestructura actual*



Nota: propia

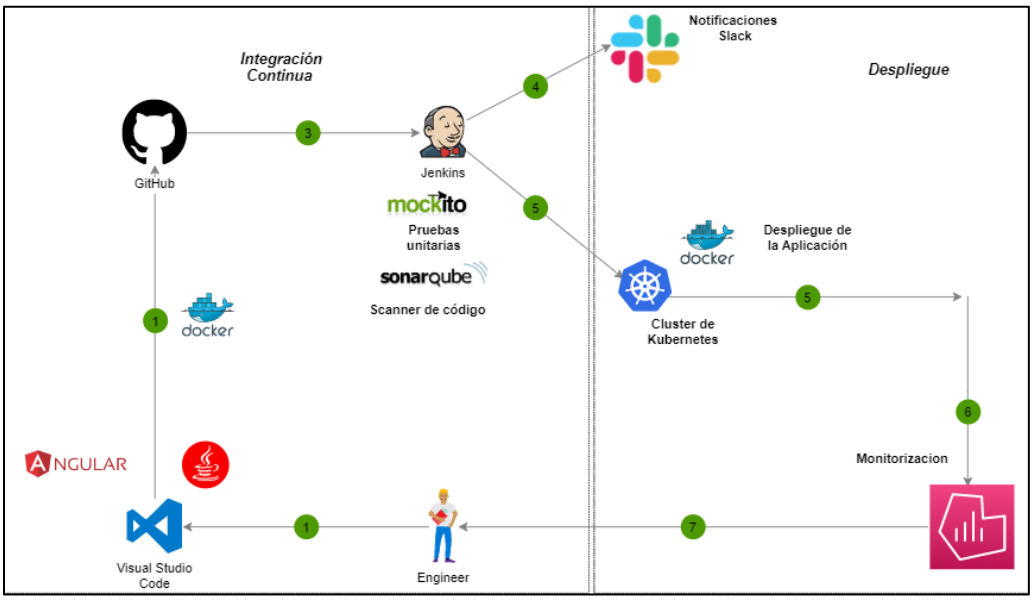
Arquitectura Propuesta.

La arquitectura propuesta consta de diferentes contenedores en donde se encuentra las herramientas seleccionadas previo análisis, estas cumplen los principios de integración y entrega continua bajo las buenas prácticas de DevOps, a través de todas sus fases. Estas fases se detallan a continuación.

- Construcción de la aplicación
- Escaneo de código
- Ejecución de pruebas unitarias
- Integración con el repositorio de código
- Despliegue automático

Adicional a los diferentes pasos que conforman la integración y entrega continua, esta arquitectura contempla notificaciones gestionadas mediante Slack herramienta la cual permite enviar notificaciones de los pipelines a los diferentes integrantes del grupo de desarrollo otra de las ventajas de esta arquitectura es el monitoreo mediante la herramienta Grafana la cual permite llevar un registro del comportamiento de la aplicación que se despliegue.

Figura 35. *Arquitectura propuesta*



Nota: En esta ilustración se muestra el paso a paso de la arquitectura propuesta.

Aplicativo y pruebas unitarias

En la ilustración 36 se muestra el desarrollo del microservicio con sus diferentes métodos que a su vez será desplegado automáticamente utilizando las buenas prácticas DevOps.

Figura 36.

Microservicio desarrollado en Spring Boot

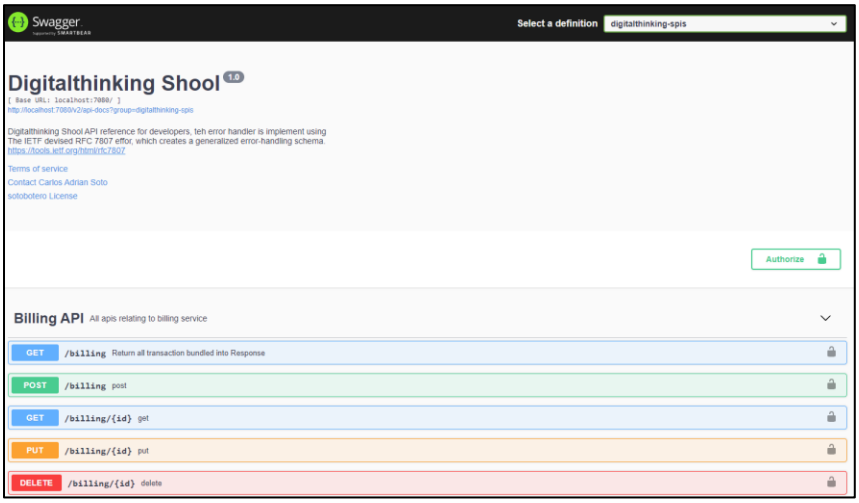


Figura 37.

Pruebas unitarias

En esta ilustración número 37 se muestra cómo se ejecuta las pruebas unitarias. En este microservicio utiliza las siguientes herramientas: SpringTest, Mockito y Junit.

Integración continua

En este apartado se muestra la ejecución de los pipelines, que a su vez están conformados por los siguientes pasos detallados en la tabla 16:

Tabla 16.

Pasos de integración continua

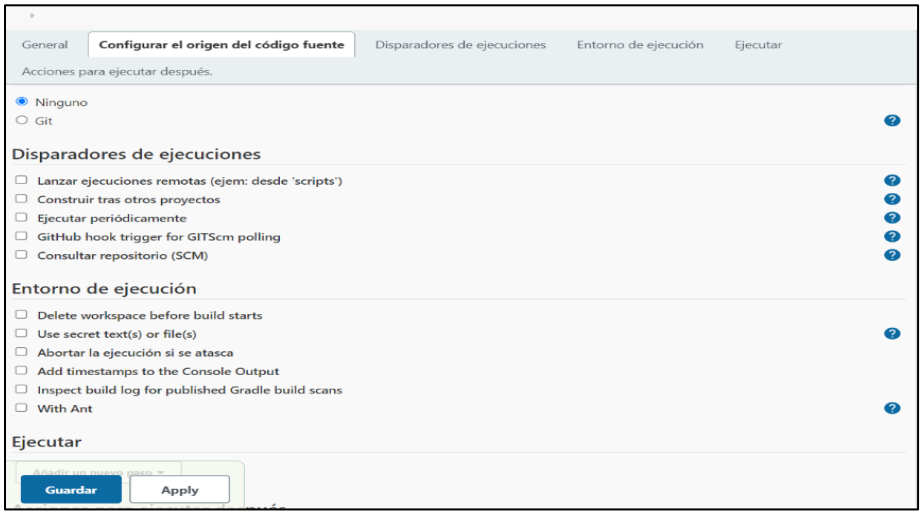
Paso	Descripción
Construcción de la aplicación	Comandos habituales, mvn clean, mvn install
Descarga de código del repositorio	Git branch, Git checkout master, git merge
Escaneo de código	Acción ejecutada por el plugin de sonarqube
Notificación de ejecución del pipeline, mediante Slack	Acción ejecutada por el plugin de Slack

Ejecución del pipeline y pruebas unitarias

En la siguiente ilustración se muestra cómo se ejecuta cada paso que contiene el pipeline.

Figura 38.

Pipeline Jenkins



En este punto se completa la parte de integración continua. El servidor de integración continua se conecta con el repositorio de código y al momento de que un programador realice el push al repositorio se ejecuta automáticamente el pipeline obteniendo el siguiente resultado mostrado en la ilustración 39.

Figura 39.

Salida de consola, pipeline exitoso

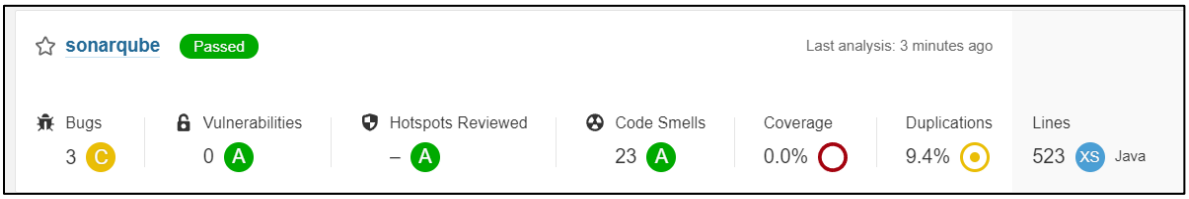
Escaneo de código y notificaciones

Para el análisis del código se instaló sonarqube que presenta varias métricas las cuales si el código desarrollado no las cumple el pipeline se rompe y no se ejecutan las demás tareas.

Al tener incorporado el Scanner de código en el proyecto al ejecutar el pipeline se obtiene un porcentaje de errores el cual está configurado para que sea menor al 80% a continuación se detalla los resultados luego del análisis de código.

Figura 40.

Resultados de análisis de código Sonarqube



Para cumplir con las buenas prácticas DevOps es necesario la instalación de una herramienta que permita notificar el estado de un pipeline a los diferente involucrados en el proceso.

La herramienta a utilizar en este paso es Slack ya que permite la comunicación del equipo involucrado en la ilustración 41 se muestra como se integra con Jenkins y notifica cada accion que se realiza en n pipeline.

Figura 41.

Integración de Notificaciones Jenkins

Despliegue continuo

El despliegue esta orquestado bajo el clúster de Kubernetes para ello es necesario la utilización de los siguientes comandos:

Tabla 17.

Comandos clúster de Kubernetes

Comando	Descripción
kubectl config view	Muestra la vista de configuraciones del clúster
kubectl --namespace default get serviceaccount	Ver el detalle de la cuenta del servidor
kubectl --namespace default get serviceaccount jenkins -o yaml	Obtener el token de la cuenta del servidor

A continuación, se crea un pipeline declarativo el cual contiene todas las instrucciones necesarias para desplegar el aplicativo en el clúster. Para ello es necesario otro proyecto el cual está compuesto por los siguientes archivos.

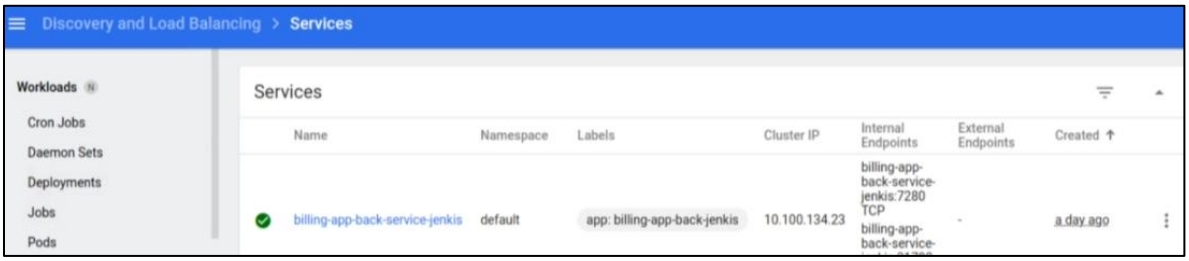
- JenkisFile
- Jenkins-account.ymal
- Deployment-Billing-app.ymal

Para mayor detalle en los scripts codificados (ver anexo 1)

A continuación, se obtiene un detalle de despliegue de la aplicación de manera automática.

Figura 42.

Despliegue automático exitoso



Análisis

Para el análisis de los resultados obtenido con la aplicación de las dos arquitecturas se basó en las siguientes dimensiones e indicadores, los que se detallan en la tabla 18.

Primero se realizó el análisis de la arquitectura actual de la empresa. En la cual se obtuvo los siguientes resultados:

DD= Días de Desarrollo

DT= Días de Testing

DP= Días de despliegue a producción

Ver anexo de recolección de datos

Tabla 18. Indicadores para arquitectura actual para el proyecto BillingApp

Dimensión	**Indicadores**	**Unidad de Medida**	**Fórmula**
Velocidad	Ciclo de vida	Días	CT = DD+DT+DP CT= Ciclo de vida CT= 23+3+1=27 Días
	Frecuencia de liberación de Código	Número	CD=10 iteraciones (Por Sprint)
Calidad	Ratio de Éxito	Porcentaje	RE = DE/TD*100% RE= 7/11*100%=63.63%

			TD = DE+DF TD=7+4=11

Análisis de indicadores arquitectura implementada

Tabla 19.

Indicadores de la arquitectura implementada para el proyecto BillingApp

Dimensión	**Indicadores**	**Unidad de Medida**	**Fórmula**
Velocidad	Ciclo de vida	Días	CT = DD+DT+DP CT= Ciclo de vida CT= 23+1/8+1/16=23 Días
	Frecuencia de liberación de Código	Número	CD=15 iteraciones (Por Sprint)
Calidad	Ratio de Éxito	Porcentaje	RE = DE/TD*100% RE= 24/25*100%=96.00% TD = DE+DF TD=24+1=25

Análisis de las arquitecturas

Los tiempos entre ambas arquitecturas se resumen en la tabla 20, la cual se obtienen los tiempos de ciclo de vida, frecuencia de liberación de código, ratio de éxito, instalación de aplicaciones y tiempo de capacitación al personal.

Tabla 20.

Indicadores entres arquitecturas

Indicadores	**Arquitectura basada en contenedores**	**Arquitectura cliente servidor**
Ciclo de vida	23 días	27 días
Frecuencia de liberación de Código	24 iteraciones (Por Sprint)	10 (Por Sprint)
Ratio de Éxito	96.00%	63.63%
Instalación de aplicaciones	117.35 s (1.95 min)	46.42 min (2785.2 s)
Tiempo de capacitación del personal	2 días	10 días

Los tiempos tomados en la arquitectura actual y de la arquitectura propuesta abarcan desde la instalación de las herramientas hasta el despliegue de una aplicación web el cual se refleja en las siguientes tablas.

Para cada proceso se necesita de un tiempo de realización, pero el análisis se focalizó la fase de instalación de herramientas, pruebas y despliegue, ya que la fase de desarrollo es variable depende de la complejidad de cada proyecto.

Capítulo V

Conclusiones y recomendaciones

Conclusiones

- El presente trabajo permitió concluir que los estudios y análisis realizados permite comprender de manera óptima el proceso de identificación de tecnologías y métodos necesarios para la implementación de la arquitectura propuesta.
- La implementación de los procesos propuestos en esta arquitectura mejora considerablemente el despliegue de aplicaciones, además de mejorar la calidad de código y la frecuencia de liberación de este.
- De acuerdo con el análisis realizado entre las arquitecturas se muestra un porcentaje del 96.00% en cuanto al indicador de calidad ya que al tener automatizado los procesos, hace que haya mayor confiabilidad por parte del programador para realizar liberación de código esto comprado con el 63.63% que se obtuvo en la arquitectura antigua.
- En este proyecto mejora considerablemente el ciclo de vida del software ya que el quipo ahorra 32 horas de trabajo, porque los procesos de prueba y despliegue se encuentran automatizados.
- DevOps permite la integración de varias funcionalidades que se ajusten a las necesidades del proyecto lo cual hace que sea más fácil escalar en cualquier entorno.
- El monitoreo implementado en esta arquitectura permite hacer revisiones constantes sin necesidad de ingresar al servidor de integración lo cual hace que sea más dinámica la detección de errores logrando así una rápida respuesta a los incidentes que ocurran durante el despliegue de la aplicación.

Recomendaciones

- Se recomienda que la arquitectura propuesta sea considerada por otras organizaciones ya que cumple con todas las buenas prácticas DevOps en la cual se refleja un mejor rendimiento en cuanto integración y despliegue.
- Al adoptar esta arquitectura se recomienda implementar un plan de capacitación en el conjunto de tecnologías planteadas como en los principios y buenas prácticas DevOps.

Limitaciones

En esta sección, se describe las limitaciones que tiene esta arquitectura, como se puede observar para la implementación de esta arquitectura se subordina todo al paso anterior. Por ejemplo, al no cumplir el umbral de código permitido que es del 80% en la fase de escaneo no continua con el siguiente que es las pruebas unitarias, en el caso de no cumplir con las pruebas unitarias de igual manera hace que el pipeline se rompa por lo que retrasa la entrega en otras fases.

Trabajos futuros

La arquitectura propuesta ha comprobado ser una arquitectura eficiente y óptima, y que puede ser replicado en otras organizaciones que busquen la automatización de los procesos del ciclo de vida del software y la eficiencia en contribución con el personal y todos los interesados, como el cliente o el usuario final. Cabe recalcar que la metodología usada para el desarrollo de la hipótesis ha dado un plus a la arquitectura, sin embargo, no es necesario utilizar la misma metodología sino la que se adapte a las necesidades de cada organización.

Adicionalmente se puede mejorar considerablemente la composición de los pipelines aumentando su escalabilidad, y que si se desea automatizar más los procesos es posible, gracias al enfoque de infraestructura como código que puede ser resuelto con grandes tecnologías como Terraform, o el enfoque de Serveless que ayuda a reducir el costo de infraestructura propia (on

premises) de cada empresa transformándolo a tener una infraestructura en un proveedor de nube.

Bibliografía

Ampalio, F. (2020). *Docker – Imágenes – Contenedores – Instancias.* Obtenido de Docker – Imágenes – Contenedores – Instancias.: https://blog.carreralinux.com.ar/2020/06/docker-imagenes-contenedores-instancias/

Balestrini, M. (2016). Como se elabora el proyecto de investigación. *BL Consul.*

BBVA. (2018). *Memoria 2018.* Obtenido de Memoria 2018.: https://extranetperu.grupobbva.pe/memoria2018/engineering.html

Betsol. (2020). *DevOps Using Jenkins, Docker, and Kubernetes*. Obtenido de DevOps Using Jenkins, Docker, and Kubernetes: https://www.betsol.com/blog/devops-using-jenkins-docker-and-kubernetes/

Bocchio, F. (2015). *researchgate.* Obtenido de Estudio Comparativo de Plataformas Cloud Computing para Arquitecturas SOA: https://www.researchgate.net/figure/Arquitectura-de-3-capas-alineada-a-SOA-Adaptado-de-70-En-las-partes-3-y-4-de-la_fig1_284348516

Campos, Y. (2018). *Estilos Arquitectónicos*. Obtenido de Estilos Arquitectónicos: http://estilosarquitectonicos.blogspot.com/

De la Torre, C. (2016). Containerized Docker Application Lifecycle with Microsoft Platform.

Deloitte. (2019). *DevOps una metodología que trabaja en el desarrollo de software ágiles*. Obtenido de DevOps una metodología que trabaja en el desarrollo de software ágiles: https://www2.deloitte.com/es/es/pages/technology/articles/devops-metodologia-desarrollo-software.html

DWS Serban Group. (2020). *DWS*. Obtenido de DWS: https://dws.gruposerban.com/blog/comparativa-aws-azure-y-google-cloud

Erl, T. (2014). *SOA Design Patterns.* Prentice Hall.

Farcic, V. (2017). The DevOps 2.0 Toolkit Automating the Continuous Deployment Pipeline. *Leanpub book*.

Forsgren, N. H. (2018). Accelerate, The Science behind of DevOps. *Portland: IT Revolution Press.*

Fowler, M. (2014). *Microservicios*. Obtenido de Microservicios: https://martinfowler.com/articles/microservices.html

García, V. (2019). *Patrón de diseño MVC*. Obtenido de Patrón de diseño MVC: https://blog.nearsoftjobs.com/patr%C3%B3n-de-dise%C3%B1o-mvc-2366948b5fc7

Gartner. (2017). *IT Glosary*. Obtenido de Gartner: https://www.gartner.com/en/information-technology/glossary/devops

Google. (2016). *Site Reliability Engineering: How Google Runs Production Systems.*

Gutierrez, D. (2015). *Estilos Arquitectónicos*. Obtenido de Estilos Arquitectónicos: https://es.slideshare.net/piojosnos/clase-08a-estilosarquitectonicos

Hernández Sampieri, R., Fernández Collado, C., & Baptista Lucio, P. (2014). *Definiciones de los enfoques cuantitativo y cualitativo, sus similitudes y diferencias.* Obtenido de Metodología de La Investigación.

Humble, J., & Farley, D. (2016). *Continuos Delivery.* I. Pearson Education, Ed.

IBM Corp. (2015). *Microservices from Theory to Practice Creating Applications in IBM.* NY: Redbooks.

ISO/IEC/IEEE. (2016). *Systems and software engineering - Architecture description.* Switerland.

Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., & Leaf, D. (2011). NIST Cloud Computing Reference Architecture Recommendations of the National Institute of Standards.

Manifesto for Agile Software. (2018). *Principios del Manifiesto Ágil.* Obtenido de Principios del Manifiesto Ágil.: https://solvingadhoc.com/los-principios-del-manifiesto-agil-y-su-aplicacion/

Matsumura, M., Brauel, B., & Shah, j. (2009). Adopción de SOA para Dummies. *Nueva Jersey: Wiley Publishing*.

McCarthy, R. (2020). *Agile y Scrum: Descubra el poder de la gestión de proyectos Agile, Lean Thinking, el proceso Kanban y Scrum.* Michigan: Pearson Education, Inc.

Menzel, G. (2015). DevOps - The Future of Application Lifecycle Automation. *DevOps*.

Microsoft Corporation. (2016). *Arquitectura de aplicaciones de .NET: Diseño de aplicaciones*. Obtenido de Arquitectura de aplicaciones de .NET: Diseño de aplicaciones: https://msdn.microsoft.com/es-es/library/ms954595.aspx

Microsoft Corporation. (2019). *Estilo de arquitectura de microservicios*. Obtenido de Estilo de arquitectura de microservicios: https://docs.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices

Microsoft, C. (2009). *Microsoft Application Architecture Guide.*

Orjuela, M., Silva, G., & Castillo, D. (2019). *Arquitecturas de software, Guía de Arquitectura de soluciones tecnológicas.*

Platzi. (2018). *Arquitectura de software*. Obtenido de Arquitectura de software: https://platzi.com/blog/que-es-arquitectura-de-software/

Pressman. (2015). *Ingeniería del Software. Un enfoque práctivo. 8va.ed.* McGraw-HIill Interamericana Editores, S.A.

Puppet. (2017). *State of DevOps Report*. Obtenido de State of DevOps Report: https://www.puppet.com/state-of-devops-report

Ravichandran, A., Taylor, K., & Waterhouse, P. (2016). DevOps for Digital Leaders. Reignite Business with a Modern DevOps – Enabled Software Factory. *USA: CA Technologies*.

Satpathy, T. (2018). *Una guía para el CUERPO DE CONOCIMIENTO DE SCRUM (Guía.* Obtenido de Una guía para el CUERPO DE CONOCIMIENTO DE SCRUM (Guía: www.scrumstudy.com

Sharma, S., & Coyne, B. (2015). *DevOps for Dummies.* Hoboken, Nueva.

Software Engineering Institute. (2010). *CMMI para Desarrollo, Versión 1.3.* Carnegie Mellon University.

SoftwareCamp. (2015). *Administración Ágil de Proyectos*. Obtenido de Administración Ágil de Proyectos: http://softwarecamp.mx/capacitacion/proyectos-atrasados-o-fuera-de-presupuesto-porque-funciona-scrum/

Suaréz, P. (2011). *Curso de Metodología de la Investigación. Población de estudio y* . Obtenido de Curso de Metodología de la Investigación. Población de estudio y : http://udocente.sespa.princast.es/documentos/Metodologia_Investigacion/Presen

Swartout, P. (2014). *Continuous Delivery and DevOps – A Quickstart Guide.* Packt Publishing Ltd.