EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Tecnológico Nacional de México

Centro Nacional de Investigación y Desarrollo Tecnológico

Tesis de Maestría

Propuesta de una arquitectura de referencia académica para la enseñanza de DevOps

presentada por

Ing. Yazmin Valeria Morales Martínez

como requisito para la obtención del grado de

Maestro en Ciencias de la Computación

Director de tesis

Dra. Blanca Dina Valenzuela Robles

Cuernavaca, Morelos, México. Octubre de 2023.

Centro Nacional de Investigación
y Desarrollo Tecnológico
Departamento de Ciencias Computacionales

Cuernavaca, Mor., 05/octubre/2023

OFICIO No. DCC/180/2023
Asunto: Aceptación de documento de tesis
CENIDET-AC-004-M14-OFICIO

CARLOS MANUEL ASTORGA ZARAGOZA
SUBDIRECTOR ACADÉMICO
PRESENTE

Por este conducto, los integrantes de Comité Tutorial de YAZMIN VALERIA MORALES MARTÍNEZ con número de control M21CE063, de la Maestría en Ciencias de la Computación, le informamos que hemos revisado el trabajo de tesis de grado titulado **"PROPUESTA DE UNA ARQUITECTURA DE REFERENCIA ACADÉMICA PARA LA ENSEÑANZA DE DEVOPS"** y hemos encontrado que se han atendido todas las observaciones que se le indicaron, por lo que hemos acordado aceptar el documento de tesis y le solicitamos la autorización de impresión definitiva.

BLANCA DINA VALENZUELA ROBLES
Directora de tesis

JUAN CARLOS ROJAS PÉREZ
Revisor 1

RENÉ SANTAOLAYA SALGADO
Revisor 2

C.c.p. Depto. Servicios Escolares.
Expediente / Estudiante

EDUCACIÓN | TECNOLÓGICO NACIONAL DE MÉXICO
09 OCT 2023
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
SERVICIOS ESCOLARES
RECIBIDO

Cuernavaca, Mor., 11/octubre/2023
No. De Oficio: SAC/164/2023
Asunto: Autorización de impresión de tesis

YAZMIN VALERIA MORALES MARTÍNEZ
CANDIDATA AL GRADO DE MAESTRA EN CIENCIAS DE LA COMPUTACIÓN
P R E S E N T E

Por este conducto, tengo el agrado de comunicarle que el Comité Tutorial asignado a su trabajo de tesis titulado **"PROPUESTA DE UNA ARQUITECTURA DE REFERENCIA ACADÉMICA PARA LA ENSEÑANZA DE DEVOPS"**, ha informado a esta Subdirección Académica, que están de acuerdo con el trabajo presentado. Por lo anterior, se le autoriza a que proceda con la impresión definitiva de su trabajo de tesis.

Esperando que el logro del mismo sea acorde con sus aspiraciones profesionales, reciba un cordial saludo.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
"Conocimiento y tecnología al servicio de México"

EDUCACIÓN | TECNOLÓGICO NACIONAL DE MÉXICO
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
SUBDIRECCIÓN ACADÉMICA

CARLOS MANUEL ASTORGA ZARAGOZA
SUBDIRECTOR ACADÉMICO

C. c. p. Departamento de Ciencias Computacionales
Departamento de Servicios Escolares

CMAZ/lmz

Agradecimientos

En primer lugar, quiero agradecer a Dios por haberme dado la fortaleza, la sabiduría y la perseverancia para enfrentar los desafíos que encontré en mi etapa de maestría. Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico brindado durante mis estudios de posgrado. Al Tecnológico Nacional de México/Centro Nacional de Investigación y Desarrollo Tecnológico por la oportunidad de realizar mis estudios de posgrado y continuar con mi formación profesional.

A mi directora de tesis, Dra. Blanca Dina Valenzuela Robles, quiero agradecerle todo su apoyo, su tiempo, su paciencia y su disposición constante para asesorarme y compartir sus conocimientos, los cuales fueron de vital importancia para concluir satisfactoriamente este trabajo. Gracias por la confianza y por contribuir en gran medida a mi formación profesional.

Al Dr. René Santaolaya Salgado, le agradezco su tiempo dedicado en la revisión de esta tesis y por sus valiosas aportaciones que me ayudaron a mejorarla a lo largo de su desarrollo. También les agradezco haber sido parte de mi formación durante mis estudios de posgrado.

A mis queridos padres, María Adela Martínez Velázquez y Ramón Morales Camacho, no existen palabras suficientes para expresar la gratitud que siento hacia ustedes. Desde el momento en que pronuncié mis primeras palabras hasta ahora, han sido mi mayor apoyo, animándome en cada paso de mi camino. Su dedicación, sacrificio y amor incondicional han sido la base sólida sobre la cual he construido mi éxito académico.
Y a mi querido hermano, Juan Francisco Morales Martínez, quiero agradecerte por ser mi compañero de vida y mi mejor amigo. Tus palabras de aliento, tu apoyo incondicional y tu presencia constante me han dado fuerzas para seguir adelante incluso en los momentos más difíciles. Juntos hemos compartido risas, lágrimas y celebraciones, y cada uno de esos momentos ha sido invaluable para mí. Gracias por estar a mi lado, por brindarme tu amor y por inspirarme a ser siempre lo mejor que puedo ser.

Este logro no solo es mío, sino también de ustedes. Cada oración, cada palabra de aliento y cada sacrificio que han hecho ha dejado una huella profunda en mi corazón y ha sido el motor impulsor detrás de mi éxito. Sin su amor, apoyo y orientación, no habría llegado hasta aquí.

Una vez más, gracias a Dios, a mis amados padres y a mi querido hermano. Que esta tesis de maestría sea un testimonio duradero de nuestro amor, fe y unidad. Sin ustedes, este logro no tendría el mismo significado. Estoy llena de gratitud y bendiciones por tenerlos en mi vida.

Con amor y agradecimiento eternos.

Agradecimientos

En primer lugar, quiero agradecer a Dios por haberme dado la fortaleza, la sabiduría y la perseverancia para enfrentar los desafíos que encontré en mi etapa de maestría. Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico brindado durante mis estudios de posgrado. Al Tecnológico Nacional de México/Centro Nacional de Investigación y Desarrollo Tecnológico por la oportunidad de realizar mis estudios de posgrado y continuar con mi formación profesional.

A mi directora de tesis, Dra. Blanca Dina Valenzuela Robles, quiero agradecerle todo su apoyo, su tiempo, su paciencia y su disposición constante para asesorarme y compartir sus conocimientos, los cuales fueron de vital importancia para concluir satisfactoriamente este trabajo. Gracias por la confianza y por contribuir en gran medida a mi formación profesional.

Al Dr. René Santaolaya Salgado, le agradezco su tiempo dedicado en la revisión de esta tesis y por sus valiosas aportaciones que me ayudaron a mejorarla a lo largo de su desarrollo. También les agradezco haber sido parte de mi formación durante mis estudios de posgrado.

A mis queridos padres, María Adela Martínez Velázquez y Ramón Morales Camacho, no existen palabras suficientes para expresar la gratitud que siento hacia ustedes. Desde el momento en que pronuncié mis primeras palabras hasta ahora, han sido mi mayor apoyo, animándome en cada paso de mi camino. Su dedicación, sacrificio y amor incondicional han sido la base sólida sobre la cual he construido mi éxito académico.

Y a mi querido hermano, Juan Francisco Morales Martínez, quiero agradecerte por ser mi compañero de vida y mi mejor amigo. Tus palabras de aliento, tu apoyo incondicional y tu presencia constante me han dado fuerzas para seguir adelante incluso en los momentos más difíciles. Juntos hemos compartido risas, lágrimas y celebraciones, y cada uno de esos momentos ha sido invaluable para mí. Gracias por estar a mi lado, por brindarme tu amor y por inspirarme a ser siempre lo mejor que puedo ser.

Este logro no solo es mío, sino también de ustedes. Cada oración, cada palabra de aliento y cada sacrificio que han hecho ha dejado una huella profunda en mi corazón y ha sido el motor impulsor detrás de mi éxito. Sin su amor, apoyo y orientación, no habría llegado hasta aquí.

Una vez más, gracias a Dios, a mis amados padres y a mi querido hermano. Que esta tesis de maestría sea un testimonio duradero de nuestro amor, fe y unidad. Sin ustedes, este logro no tendría el mismo significado. Estoy llena de gratitud y bendiciones por tenerlos en mi vida.

Con amor y agradecimiento eternos.

Abstract

Currently, software development has become a highly competitive and demanding task in terms of quality and continuous delivery. To achieve greater efficiency at work and reach goals in less time, companies have implemented enhanced practices that adapt to their needs. One of these practices is DevOps, which seeks to unite the teams involved in a software development project and eliminate barriers that prevent working together. DevOps is not just a set of automation tools, but a culture that fosters collaboration and continuous development. World-renowned companies such as Amazon, LinkedIn, Google, Facebook, Spotify, and Netflix have adopted these practices. For successful adoption of DevOps, a cultural change in team members is necessary, as well as a good reference architecture that facilitates the path and provides a design and implementation solution for a DevOps environment. This work aims to play an incipient role in the adoption of DevOps practices for academia through the proposal of an academic reference architecture for teaching DevOps that provides a solution and guidance in the management and development of projects under the DevOps paradigm in the field.

Resumen

En la actualidad, el desarrollo de software se ha convertido en una tarea altamente competitiva y exigente en términos de calidad y entrega continua. Para lograr una mayor eficiencia en el trabajo y alcanzar los objetivos en un menor tiempo, las empresas han implementado prácticas mejoradas que se adaptan a sus necesidades. Una de estas prácticas es DevOps, que busca unir a los equipos que participan en un proyecto de desarrollo de software y eliminar las barreras que impiden trabajar coordinadamente. DevOps no es solo un conjunto de herramientas de automatización, sino una cultura que fomenta la colaboración y el desarrollo continuo. Empresas de talla mundial como Amazon, LinkedIn, Google, Facebook, Spotify y Netflix han adoptado estas prácticas. Para una adopción exitosa de DevOps, es necesario un cambio cultural en los miembros del equipo y una buena arquitectura de referencia que facilite el camino y proporcione una solución de diseño e implementación para un entorno DevOps. Este trabajo pretende jugar un papel incipiente en la adopción de prácticas DevOps para la academia a través de la propuesta de una arquitectura de referencia académica para la enseñanza de DevOps que brinde una solución y orientación en la gestión y desarrollo de proyectos bajo el paradigma DevOps en el ámbito de la Ingeniería de Software.

Contenido

Índice de figuras

Índice de tablas

Capítulo I

1. Introducción

En la actualidad el desarrollo de software, principalmente el gestionado en la industria (nacional e internacional), ha sido siempre una tarea de carácter competitivo, de exigentes y altos estándares de calidad, demandantes de una entrega continua (CD) limpia y productos integrados (CI)[1] y a fin de lograr sus objetivos en un menor tiempo de desarrollo que aceleren su productividad han creado y adaptado nuevas y mejoradas prácticas que atiendan sus necesidades.

Todas estas cualidades requeridas en la industria, se consolidaron en el 2009 [2] en una serie de prácticas denominadas DevOps, que nacen de la experiencia y necesidad de comprender las habilidades y destrezas que conducen a la excelencia en la entrega de tecnología con poderosos resultados empresariales [3], [4].

DevOps surge a partir de una necesidad y un objetivo, el cual es unir a los equipos o personas que participan en un proyecto de desarrollo de software. DevOps no es solo un conjunto de herramientas de automatización, procesos iterativos o determinadas prácticas; implica personas, valores de colaboración, de tal manera que se considera una cultura o filosofía que busca eliminar las barreras o silos, creados entre equipos, que no permiten trabajar coordinadamente y de forma homogénea, por la falta de comunicación [5].

Básicamente, DevOps promete la integración continua, el despliegue continuo, entrega rápida y automatizada de software en pequeñas versiones, además de gestionar el surgimiento de defectos de forma eficiente y mejorar la resiliencia. Dentro de un entorno DevOps los desarrolladores se encargan de liberar el software de manera frecuente, mientras que el equipo de operaciones protege la estabilidad de la infraestructura [5].

Estas atractivas ventajas hacen que empresas de talla mundial como *Amazon , LinkedIn , Google*, *Facebook, Spotify y Netflix* sean partícipes del uso de las prácticas de DevOps [6].

Pero, ¿Cómo se lleva a cabo la adopción de estas prácticas de DevOps?. Muchos expertos señalan, comenzar por un cambio cultural en los miembros de un equipo, agregar nuevas tecnologías, infundir colaboración y desarrollo, una buena arquitectura de referencia[6].

Así que, en el reto de una adopción exitosa de DevOps una arquitectura de referencia puede facilitar el camino y proporcionar una solución de diseño e implementación para un entorno DevOps con sus prácticas y procesos que permiten la automatización y la integración continua del despliegue de aplicaciones de software [7].

Por lo tanto, este trabajo pretende jugar un papel incipiente en la enseñanza de prácticas DevOps para la academia a través de una arquitectura de referencia que brinde una solución y soporte en la gestión de proyectos de Ingeniería de Software, área en la que se han detectado proyectos inconclusos por falta de seguimiento, control y registro, comúnmente descartados por falta de configuración e inconsistencias en su infraestructura.

1.1. Planteamiento del problema

La industria del desarrollo de software ha enfrentado a lo largo de los años diversos desafíos en cuanto a la creación y entrega de productos de alta calidad. Con el tiempo, se han adoptado distintos enfoques con el propósito de mejorar tanto la eficiencia como la calidad de los productos resultantes. Las metodologías y herramientas de desarrollo han sido pilares fundamentales en esta evolución.

La introducción de prácticas como la integración continua y la entrega continua ha permitido abordar algunos de los problemas identificados en el desarrollo de software. A pesar de que los equipos de trabajo en diferentes departamentos comparten un objetivo común, la falta de coordinación a menudo provoca problemas al integrar y desplegar el producto final.

En el año 2010, surge un nuevo paradigma conocido como DevOps, que aborda precisamente las problemáticas que surgen entre los equipos de desarrollo y operaciones. Este enfoque ha ganado reconocimiento en la industria de Tecnologías de la Información (TI), y a partir de entonces, empresas como Amazon, LinkedIn, Google, Facebook, Spotify y Netflix han reconocido cómo la integración entre desarrollo y operaciones puede mejorar significativamente la eficiencia y calidad del software.

A pesar de la adopción del paradigma DevOps, no todas las industrias han logrado llevar a cabo la transición en su ecosistema de desarrollo. Esto se debe a que la transición implica no solo la implementación de herramientas, sino también un cambio cultural que promueva la coordinación y la comunicación entre los equipos.

Además, en [8]–[11] se reconoce la creciente demanda de profesionales que posean conocimientos en DevOps y las habilidades necesarias para implementar esta práctica en las industrias. Sin embargo, se ha identificado una brecha en la enseñanza del paradigma DevOps en los programas educativos [12], [13]. Actualmente, no existen estrategias que faciliten la identificación de la información general del dominio DevOps con mayor claridad, sin concentrarse en herramientas que se desvíen del propósito de aprender el paradigma. Es necesario contar con un patrón genérico en la enseñanza de DevOps en la academia para su integración rápida y estructurada en los planes de estudio [8], [14].

Tomando en cuenta lo anterior, es posible observar que las investigaciones del estado del arte de la ingeniería de software con respecto a la enseñanza de DevOps resaltan varios desafíos en la industria del desarrollo, sin embargo, se destaca la escasez de profesionales con conocimientos en DevOps cuya principal causa es el rezago en la enseñanza de este paradigma en los programas educativos. A su vez, se reconoce la complejidad inherente a la enseñanza y el aprendizaje de DevOps, por lo tanto, existe la necesidad y la oportunidad de establecer estrategias facilitadoras y de un enfoque general para la enseñanza de DevOps.

1.2. Objetivos

Objetivo general

Identificar los principios y procesos clave del paradigma DevOps e integrarlos en una estructura básica conceptual de orientación académica, proponiendo una Arquitectura de Referencia como guía en el desarrollo e implementación de proyectos de software a través de los principios y procesos identificados.

Objetivos específicos

- Analizar y seleccionar un modelo o proceso de desarrollo para la arquitectura de referencia académica.
- Generar un esquema gráfico conceptual que permita visualizar los procesos clave de DevOps en el desarrollo y entrega de software.
- Generar una guía de la arquitectura de referencia académica.

1.3. Justificación

Esta investigación propone una Arquitectura de Referencia de orientación académica que aborda algunos de los problemas identificados en el estado del arte. En primer lugar, se reconoce la complejidad inherente a la enseñanza y el aprendizaje de DevOps, así como la falta de herramientas y estrategias pedagógicas que permitan una comprensión clara de la aplicación de DevOps.

También es fundamental superar la tendencia de centrarse únicamente en las herramientas y desviar el enfoque principal de comprender el paradigma DevOps. La Arquitectura de Referencia propuesta en esta tesis tiene como objetivo proporcionar una visión conceptual integral del dominio DevOps, permitiendo a los estudiantes identificar la información básica necesaria para su comprensión.

Además, esta Arquitectura de Referencia pretende servir como una guía o patrón genérico en el desarrollo y la enseñanza de DevOps, proporcionando a los estudiantes la oportunidad de practicar con proyectos académicos o de investigación. La arquitectura incorpora el estándar IEEE 2675:2021 [15] con lo que pretende agregar calidad a los trabajos a partir del uso de este estándar, preparando así a los alumnos para los desafíos del mundo laboral.

En resumen, la propuesta de una Arquitectura de Referencia de orientación académica pretende abordar algunas de las deficiencias actuales en la enseñanza de DevOps, proporcionando una base sólida para la comprensión conceptual y la práctica. Al integrar estándares y orientación en la enseñanza de DevOps, esta propuesta busca impulsar el éxito y desarrollo profesional en este campo.

1.4. Alcances y limitaciones

Alcances

- Desarrollo de una arquitectura de referencia de DevOps para un entorno académico.
- Configuración e integración de herramientas DevOps.

- Las prácticas que se considerarán especialmente son: despliegue continuo, entrega continua, test, codificación, compilación, planificación, versionado, despliegue, operaciones, compilación, pruebas continuas, monitoreo, automatización.
- Integración de conceptos básicos de DevOps.
- Arquitectura de referencia de orientación académica.
- Desarrollo de guía en el desarrollo e implementación de proyectos de software.
- Prácticas DevOps técnicas.
- Validación cualitativa con personas con experiencia en DevOps, arquitecturas de referencia, académicos e investigadores.

Limitaciones

Quedan fuera del alcance de este trabajo de tesis los siguientes puntos:

- Servidores en la nube.
- La selección de herramientas queda a criterio libre e independiente de cada usuario.
- Medir la calidad de software.
- Medir la calidad de los procesos.
- Seguridad básica (DevSecOps).
- Prácticas DevOps no técnicas.

Capítulo II

2. Marco conceptual

El presente capítulo aborda los conceptos teóricos requeridos para la comprensión de este trabajo.

2.1. Arquitecturas de referencia

Según [16] una arquitectura de referencia (AR) se define como un *"referente"* para los equipos involucrados en el desarrollo de sistemas de software, que proporciona conocimientos sobre cómo diseñar arquitecturas de software en un dominio técnico o aplicación específica [17], [18].

De acuerdo con [19], una AR es un tipo de arquitectura genérica y abstracta que incluye estándares y legislaciones, conocimiento del dominio, elementos de software y hardware, estilos y patrones arquitectónicos, y las mejores prácticas de desarrollo de software, entre otros artefactos. Las arquitecturas de referencia también proporcionan plantillas y directrices para diseñar sistemas de software en un dominio concreto. Por lo tanto, de acuerdo con [20] el propósito de una AR es guiar el desarrollo, estandarización y evolución de sistemas, a la vez en [21] se menciona que, el uso de una AR mejora la reutilización y reduce el tiempo empleado durante las actividades de software [19]. Las AR han apoyado con éxito el desarrollo de sistemas críticos en la industria en dominios como automoción, sanidad, industria 4.0 y la agricultura [22]. Una AR se puede establecer para la estandarización de arquitecturas concretas o para facilitar el diseño de dichas arquitecturas. En este sentido, una AR debe ser descrita para comunicar de forma fiable el conocimiento que contiene. Las descripciones de arquitecturas se utilizan para mejorar la comunicación y la cooperación de los equipos de software, permitiéndoles trabajar de forma integrada y coherente, mientras que se mejora el desarrollo de software a través de la reutilización y la estandarización [23].

Enfoques para describir arquitecturas de referencia.

Según un estudio realizado en 2020 [22], uno de los procedimientos para describir una Arquitectura de Referencia (AR) es la "Orientación de los enfoques". Se propone partir del contexto en el que se propusieron y validaron las arquitecturas, ya sea en un entorno académico o industrial. El mismo estudio identificó ocho tipos de enfoques para apoyar la descripción de las AR, por: 1) procesos, 2) métodos, 3) lenguajes de descripción arquitectónica (ADL), 4) estilo arquitectónico, 5) marco, 6) modelo, 7) punto de vista y 8) plantilla de documento.

Cada uno de estos enfoques proporciona una forma de reutilizar las decisiones de diseño y restricciones que se aplican a una Arquitectura de Referencia (AR) para inducir cualidades deseables elegidas [22]. Para efectos de esta investigación, se considerarán dos enfoques en particular: el enfoque por procesos, que define lo que hay que hacer sin especificar cómo debe realizarse cada tarea, y el enfoque por modelo, que pretende mejorar la comprensión de lo que son las AR, sus componentes y relaciones, ayudando al establecimiento, uso y evolución

de dichas arquitecturas. Un ejemplo de este último enfoque por modelo es RAModel [24] (Reference Architecture Model), un modelo de referencia para AR que puede servir como base para otras arquitecturas de referencia.

Otro enfoque dentro de [22] son las *vistas arquitectónicas y puntos de vista*: una vista arquitectónica expresa la arquitectura de un sistema desde la perspectiva de un interés específico del sistema. Un punto de vista de arquitectura establece las convenciones para la construcción, interpretación y uso de vistas de arquitectura para enmarcar preocupaciones específicas del sistema [25].

En [22] se hace referencia a veinticuatro vistas, estas son: vista conceptual, vista lógica funcional, vista de proceso, vista de componentes, vista de implementación , vista de escenario, vista de plataforma, vista técnica, vista física, vista, vista de despliegue, vista de modelo, vista de contexto, vista informal, vista de modelos de información, vista de dominio, nueva vista, vista de interfaz, vista de código, vista de módulo, vista de ejecución, vista de ejecución, nueva vista, vista filtrada y vista aumentada.

En [25] se menciona que, una vista conceptual es una representación abstracta y de alto nivel de la AR, que describe los conceptos fundamentales y las relaciones entre ellos. Esta vista se enfoca en los conceptos clave que definen la AR, como los componentes, módulos, interfaces y dependencias, y cómo se relacionan entre sí para lograr los objetivos de la AR. La vista conceptual es importante porque proporciona una comprensión común de la AR a todos los equipos involucrados en el desarrollo de software, lo que facilita la comunicación y la colaboración. Además, permite una mayor flexibilidad y adaptabilidad en el diseño de sistemas de software, ya que los equipos pueden entender mejor cómo se relacionan los diferentes componentes y cómo se pueden modificar para satisfacer las necesidades específicas del sistema.

2.1.1. Tipos de Arquitectura de referencia

Existen dos tipos de orientación para arquitecturas de referencia, académicas e industriales [22]. Ambas arquitecturas tienen su propósito y aplicaciones específicas, las académicas son más teóricas, buscan investigar y entender los conceptos y problemas. Por otro lado, las arquitecturas de la industria son más prácticas y buscan proporcionar un marco para implementar soluciones en una organización real (Ver Tabla 1. Comparativa).

Tabla 1. Arquitecturas de referencias académicas vs. industriales

	Académica	Industrial
Enfoque	Basado en investigaciones y estudios científicos.	Basada en la práctica e implementación de organizaciones reales.
Alcance	Suelen ser más teóricas y generales.	Específicas y aplicables a una situación particular.
Fiabilidad	Pueden ser más rigurosas en términos de investigación y validación.	Pueden ser más flexibles y basadas en la experiencia.
Aplicabilidad	Pueden ser difíciles de implementar debido a su complejidad y enfoque teórico.	Fáciles de implementar debido a su naturaleza práctica y la experiencia previa.

NOTA: Ambas arquitecturas tienen su propósito y aplicaciones específicas, las académicas son más teóricas, buscan investigar y entender los conceptos y problemas. Por otro lado, las arquitecturas de la industria son más prácticas y buscan proporcionar un marco para implementar soluciones en una organización real.

2.1.2. ISO/IEC/IEEE 42010:2011 Systems and software engineering—Architecture description

Según [25], el propósito principal de IEEE 42010 es definir cómo se deben describir las arquitecturas de sistemas y software y cómo se deben representar esas descripciones en términos de diferentes puntos de vista y preocupaciones. La norma también proporciona directrices para especificar los puntos de vista, los marcos de puntos de vista y las correspondencias.

En resumen, IEEE 42010 proporciona un marco y un conjunto de directrices para describir, representar y analizar arquitecturas de sistemas y software de manera coherente y estandarizada.

2.2. DevOps

El término DevOps nace en la conferencia de metodología Ágil en el 2008 en Toronto y enfatiza la necesidad de disolver la ruptura entre los equipos de desarrollo y operaciones para que puedan colaborar y responder a la necesidad de una rápida y constante entrega de software [2], [5], [6], [14].

Estas prácticas, además de promover la entrega rápida y frecuente, buscan garantizar la calidad del software entregable. Así, DevOps es una cultura que establece la colaboración entre todas sus partes implicadas en el desarrollo, el despliegue y el funcionamiento del software para ofrecer un producto de alta calidad en menor tiempo [26].

2.2.1. Filosofía DevOps

DevOps es considerado como un nuevo movimiento que busca vincular y coordinar a los miembros del equipo de desarrollo y operaciones. Para entender esta filosofía, es necesario entender las razones de su origen. Por lo general, dentro de las organizaciones estas áreas tienden a generar una brecha entre ellas por la falta de estructura de gestión entre los desarrolladores y las operaciones, aunque existen varios factores que provocan esta situación [2], [27], [28].

2.2.2. El ciclo de vida DevOps

El ciclo de vida representa los pasos o fases que abarcan los procesos de desarrollo del código fuente de un sistema de software en producción para ser desplegado en un entorno. Son ocho etapas o fases en las que los equipos de desarrollo y operaciones intervienen en el desarrollo del proyecto, además del icónico símbolo de iteración que representa las etapas de vida DevOps de una aplicación, entre estas fases podemos mencionar: planificación, codificación, construcción, pruebas, lanzamiento, operación, monitoreo y despliegue [2], [27], [29].

En [2], [29] se describen las fases del ciclo de vida DevOps como sigue:

Fase 1 - Planificación: Presenta un panorama general que orienta a los participantes acerca del motivo y meta del proyecto en cuestión. En esta etapa se definen las funcionalidades en cada iteración y los indicadores de aceptación a cumplir. Es la base entre la colaboración y la comunicación entre el equipo de negocios (operaciones) y el equipo de desarrollo. Define las tareas de cada equipo involucrado e identifica los requerimientos que debe contener el entregable [27].

Fase 2 - Codificación: Esta tarea es principalmente ejecutada por el equipo de desarrollo y es donde empiezan a codificar o escribir líneas de código para el software. La codificación progresa en pequeños procesos durante todo el proyecto (es continuo e iterativo).

De manera paralela, el equipo de operaciones, comienza a definir pruebas y construir las automatizaciones pertinentes para el software y así confirmar que cumple con las especificaciones de calidad.

Fase 3 - Construcción: Esta fase lleva a cabo la integración de varios códigos que se crearon anteriormente en la fase de codificación y una vez cumplida esa tarea es hora de crear los nuevos artefactos que componen el software, incluyendo sus

funcionalidades. Este proceso es crítico, ya que un error en la construcción de código puede provocar que deje de funcionar el proyecto.

Fase 4 - Pruebas: En este paso son ejecutadas las pruebas para corroborar el óptimo funcionamiento del código generado y el cumplimiento de las características que debe cubrir el entregable. Aquí se consideran tanto las nuevas pruebas unitarias como también las que aseguran el cumplimiento de la especificación delimitada en la planeación.

Fase 5 - Lanzamiento: Esta etapa de la vida del software integra el código desarrollado y realiza pruebas para que siga el desarrollo continuo que es posible a través de la integración y las pruebas recurrentes.

Fase 6 - Operación: Esta es la etapa, en la que el equipo de operaciones asume el control de la aplicación en producción y debe asegurarse de que no haya comportamientos extraños o inapropiados que puedan encontrar en la producción y a la vez si es necesario notificar a otros miembros del equipo el problema identificado para tomar decisiones inmediatas. También, en esta fase se prevén mejoras durante y después de la producción.

Fase 7 - Despliegue: Por último, la fase de implementación toma lo que se ha desarrollado durante el proyecto y procura ponerlo a disposición de los clientes/usuarios, después de la fase de lanzamiento (involucra la configuración de dichos entornos y estructuras).

Fase 8 - Monitoreo: Establece los parámetros a contemplar de la aplicación (tarea ejecutada por el equipo de operaciones). Esta fase recopila toda la información a lo largo de la ejecución del proyecto con el fin de ajustar (monitorizar y controlar el estado de las aplicaciones y su infraestructura) y llevar a cabo acciones necesarias.

2.2.3. Principios y prácticas DevOps

Las prácticas DevOps establecen la integración continua y despliegue continuo (CI/CD) en cada fase con el apoyo de herramientas que automatizan y agilizan las labores de los equipos involucrados en el desarrollo de código, además de procurar romper la brecha de comunicación y colaboración entre los equipos de desarrollo y operaciones [2], [28].

La organización de equipos (desarrolladores y operaciones): es un factor determinante de éxito para llevar a cabo prácticas DevOps. Disolver *silos* entre los equipos y permitir que se comuniquen y se ayuden es uno de los desafíos que resuelve esta filosofía DevOps. Qué mejor forma de comunicar a estos equipos de desarrolladores y operaciones, reuniéndose frecuentemente para que, juntos, busquen soluciones a sus problemas y colaboren en los resultados [28].

Una *Integración continua CI* consta de compilar, probar y entregar una aplicación de forma frecuente mediante las siguientes prácticas:

- Gestionar un repositorio de código fuente (Git por ejemplo).
- Construir y compilar aplicaciones de manera automática (Creación de pipelines).
- Hacer pruebas unitarias frecuentemente.
- Construir y empaquetar las aplicaciones regularmente.
- Informar al equipo de desarrollo de algún problema que surja.

Pipeline

Según, [2] la práctica de automatización como un pipeline es una serie de procesos automatizados que se utilizan para compilar, probar y entregar software de manera eficiente y confiable. Estos procesos pueden incluir la integración continua, la entrega continua y la implementación continua [30].

El despliegue continuo (CD) es una práctica de DevOps en la que los cambios de código se construyen, prueban y despliegan automáticamente en un entorno de producción o de pruebas. Esto permite a los equipos de desarrollo entregar cambios de forma rápida y segura a los usuarios finales [30].

En [27] se menciona que la filosofía DevOps denota la mejora continua de sus procesos y herramientas. El primer paso es recopilar periódicamente comentarios e indicadores. Luego, sobre la base de esta información, formalizar, definir y seguir un plan de mejora. Sin embargo, hay que tener cuidado de no hacer indicadores que puedan hacer que los promotores y las operaciones entren en problemas. Los indicadores que hay que considerar deben ser aquellos que reduzcan el tiempo de comercialización.

Otros procesos, según [27] y [31], a lo largo del ciclo de vida DevOps de una aplicación, existen otros procesos de administración y gestión, tales como:

- La monitorización y registro de datos para facilitar el diagnóstico.
- Incidentes y problemas (reuniones del personal involucrado).
- El rendimiento (trata de la integración de las pruebas de rendimiento de la Infraestructura y automatizando).
- La seguridad.
- El plan de recuperación de problemas y desastres.
- Herramientas DevOps: Algunos de los conceptos para la aplicación de herramientas son importantes para una adopción exitosa de DevOps las cuales son:
- Control de versiones.
- Herramientas de construcción y empaquetado.
- Herramientas de Integración continua.
- Infraestructura virtual.
- Gestión de la configuración.
- Orquestación y despliegue de aplicaciones.
- Monitorización.

2.2.4. Estándar IEEE 2675:2021

Las normas IEEE son documentos desarrollados por voluntarios con experiencia científica, académica e industrial en grupos de trabajo-técnico [15].

El objetivo de esta norma es especificar las prácticas necesarias para que las operaciones, el desarrollo y otras partes interesadas clave colaboren y se comuniquen para desplegar sistemas y aplicaciones de forma segura y fiable.

2.3. Metodologías ágiles

Las metodologías ágiles, establecidas en 1990, contrarias a las metodologías tradicionales, son un cambio cultural y de procedimientos en el que se trabaja con un enfoque iterativo e incremental para producir software con más rapidez, se destaca por una manera flexible y adaptativa de desenvolverse para satisfacer las necesidades del cliente [11] y [32].

Estas metodologías ágiles trabajan bajo el concepto de etapas o pasos, iniciando por la recopilación de requisitos o retroalimentación, el diseño, el desarrollo, las pruebas, el despliegue, y la retrospectiva. Estos procedimientos se llevan a cabo de manera iterativa para solucionar o mejorar un proceso de desarrollo de software. Algunos ejemplos de metodologías ágiles son: Scrum, XP, Lean, Crystal, RUP, ASD, FDD, ICONIX, AUP, Lean Software Development [32].

Capítulo III

3. Estado del Arte

En esta sección se muestra un panorama de lo que ya se ha hecho y descubierto en relación con el tema, y establece el escenario para la contribución única y original que la investigación actual pretende hacer.

3.1. Enseñanza de DevOps en la academia

Se profundizó en el tema de la enseñanza de DevOps en la educación superior para plantear esta investigación. En los últimos años, la enseñanza de DevOps en la academia ha sido un tema de interés creciente, y varios trabajos previos, como [8], [9], [12], [14], [28], [33]–[36] han abordado esta cuestión. Estos trabajos incluyen planes de estudio para Ingeniería de Software sobre la filosofía DevOps, por ejemplo, [9], [11], [34], [37]. Las investigaciones también detallan cómo las universidades han logrado involucrar e introducir a sus estudiantes en la nueva tendencia del mercado e industria para adoptar una cultura DevOps. Además, comparten sus experiencias en la enseñanza de DevOps y cómo convierten estas prácticas en una habilidad natural de desarrollar software para el alumno.

Los cursos de DevOps son una estrategia ampliamente utilizada, como [12], [36], [38], enfocados en procesos ágiles que se basan en desplegar versiones pequeñas y frecuentes mediante iteraciones, adaptarse a cambios y promover la comunicación en los equipos para generar retroalimentación constante.

Investigaciones como [8], [13], [28], [37] se centran en las nuevas exigencias de la industria hacia la academia para educar la mano de obra profesional para trabajos de Ingeniería de Software con competencias blandas como DevOps. La respuesta de la academia fue que existe un déficit en el sistema educativo para adaptarse a estas nuevas exigencias [36].

El artículo [13] aborda los desafíos de enseñar DevOps en la educación superior, con perspectivas extraídas de dieciocho investigaciones. Entre los hallazgos y recomendaciones para la enseñanza de DevOps en la academia se encuentran:

- Se identificó que la enseñanza basada en proyectos es un mecanismo exitoso.
- Se sugiere crear un ambiente relajado y de confianza entre alumnos y profesores.
- DevOps es una materia ideal para utilizar métodos no tradicionales de enseñanza, como la instrucción entre pares, el aula invertida y el aprendizaje basado en problemas.
- El método de "Aprendizaje Basado en Problemas (ABP)" es adecuado para impartir prácticas DevOps.
- El plan de estudios debe coordinar la disciplina de Arquitectura con DevOps, ya que se complementan.
- Se recomienda tomar cursos sobre teorías, herramientas y adaptación antes de incursionar en la disciplina DevOps.
- Establecer un nivel común de conocimiento para todos los diferentes grupos.

Entre los desafíos de la enseñanza de DevOps se encuentran:

- No está claro que deberían aprender para comenzar en DevOps [14].
- La necesidad de implementar estrategias que no dependan del uso de una herramienta específica para trabajar, lo que otorgaría libertad y flexibilidad a los estudiantes. Esto se diferencia de otros recursos de enseñanza que sí utilizan herramientas específicas, como se describe en las referencias [12], [31], [36], [39], [40].
- Los estudiantes tienen dificultades para explicar conceptos como la "Infraestructura como Código", que están relacionados con la operación.
- Falta de estrategias que evitan en los alumnos concentrarse en las herramientas DevOps y desvíen su propósito de aprender el paradigma.
- Estrategias que faciliten identificar información del dominio DevOps.
- La necesidad de implementar estrategias que otorguen a los estudiantes un control general para desarrollar proyectos bajo el paradigma DevOps, lo que les permitiría tener una guía y un patrón genérico para su trabajo.
- Contar con un patrón y una guía para el desarrollo de proyectos que involucren normas y estándares que otorguen calidad a los trabajos realizados.
- La necesidad de implementar una estrategia que permita una visualización general y resumida de lo que implica la aplicación de DevOps.

Estos trabajos han destacado la importancia de incorporar los principios y prácticas de DevOps en los planes de estudio de la academia. La enseñanza de DevOps puede ayudar a los estudiantes a desarrollar habilidades valiosas en la industria, como la colaboración en equipo, la automatización y el monitoreo continuo [8].
En resumen, la enseñanza de DevOps es un tema importante en la academia y ha sido abordado por varios trabajos previos. Es fundamental que las instituciones educativas sigan explorando y desarrollando sus programas y estrategias de enseñanza para preparar a los estudiantes para el mundo laboral actual.

3.2. Arquitecturas de referencias DevOps genéricas

3.2.1. A reference architecture for the container ecosystem

La investigación [41] se enfoca en los modelos arquitectónicos utilizados para representar y entender los ecosistemas y su construcción. Se describen modelos de contenedores de componentes de ecosistemas, los cuales son abstractos y generalizan los sistemas para unificar su vocabulario y guiar en su complejidad y heterogeneidad. La arquitectura de referencia mencionada, cumple con ser abstracta y genérica, además se destaca la importancia de los patrones como solución para resolver dilemas frecuentes. La ventaja de una arquitectura es que garantiza el cumplimiento, la seguridad, la fiabilidad o la gobernanza de los ecosistemas de contenedores. Los

contenedores son una solución portátil y ligera de virtualización que facilita el desarrollo, despliegue y distribución de aplicaciones en entornos informáticos. Los modelos presentados están descritos con base en sus funciones, no en los dispositivos. La investigación concluye resaltando las ventajas de iniciar una arquitectura de referencia basada en un conjunto de patrones que ayuden en la visualización y estructura e interacciones de sus componentes, añadiendo valores de calidad, seguridad y fiabilidad.

3.2.2. DevOps Reference Architecture for Multi-Cloud IOT Applications

En [42] se propone una arquitectura de referencia DevOps (DRA) para aplicaciones IoT en multi-nube, que fue desarrollada y evaluada por el método DSR (Design science research methodology for information systems research). El Modelo Conceptual DRA aquí presentado, combina la vista conceptual de conceptos DevOps y Multi-Cloud para IoT. Su objetivo es proporcionar a los equipos de proyectos DevOps una visión clara del proyecto y definir una teoría general sobre cómo debe ser la comunicación y colaboración a nivel profesional. DRA utiliza un enfoque ágil y adaptativo para la planificación y diseño continuos de proyectos. El enfoque DevOps mejora la comunicación del equipo y proporciona un mapeo más claro del proyecto, permitiendo la sincronización de ideas, esfuerzos, recursos y resultados.

En conclusión de [42] menciona que, esperan que su DRA brinden a profesionales e investigadores, información consolidada y sintetizada, lo que les permitirá tomar decisiones informadas sobre la arquitectura y la implementación sistemática de DevOps para el despliegue de IoT en multi-nube. Su trabajo podría proporcionar una guía clara y precisa para optimizar los procesos de comunicación y colaboración en los equipos de proyectos DevOps, facilitando así el éxito en la implementación de proyectos IoT en entornos de multi-nube.

3.2.3. A Contribution to the Establishment of Reference Architectures for Mobile Learning Environments

El artículo [43] presenta la propuesta de una arquitectura de referencia para entornos de aprendizaje móvil llamada Ref-mLearning. Esta arquitectura establece directrices para el desarrollo, reutilización y la interoperabilidad de los entornos de aprendizaje móvil basados en aspectos SOA. La metodología utilizada para su desarrollo fue ProSA-RA y el método de evaluación utilizado fue el RAModel. Se concluye que Ref-mLearning es completo y presenta los elementos más importantes en relación con aspectos de movilidad y SOA, lo que garantiza una mayor comodidad y flexibilidad al entorno de aprendizaje, así como una mayor reutilización, interoperabilidad y estandarización mediante el uso de servicios web.

3.2.4. CALMS de DevOps en mipymes desarrolladoras de software en el contexto surcolombiano

En el artículo [44] se presenta un conjunto de lineamientos para la implementación del modelo CALMS (DevOps) en mipymes de desarrollo de software en el contexto surcolombiano, incluyendo aspectos técnicos y organizacionales para lograr un ambiente de desarrollo novedoso que permita la integración de DevOps al contexto del desarrollo de software colombiano. El modelo fue probado en una empresa mipyme y los resultados fueron exitosos, permitiendo un despliegue diario y una mayor productividad. Además, se discute cómo DevOps puede incrementar la competitividad de las empresas de software en un mercado globalizado.

3.3. Tabla de trabajos relacionados

En la Tabla 2 se presenta una comparativa de las investigaciones más recientes en el estado del arte. Se han incluido las arquitecturas de referencia genéricas, que son adaptables a diferentes contextos y no se limitan a un caso particular de empresa. Las arquitecturas de empresas particulares que sólo atienden su caso específico han sido excluidas. El objetivo es mostrar las opciones más amplias y versátiles que pueden ser aplicadas en diferentes escenarios, incluyendo la enseñanza. De esta manera, se busca ofrecer una visión más completa y útil para aquellos que buscan implementar soluciones tecnológicas en sus proyectos.

A continuación, se describen cada uno de los criterios que se emplearon para realizar la tabla comparativa.

Arquitecturas de Referencia académicas: Nombre de la Arquitectura de referencia DevOps.
Objetivo: Su objetivo de desarrollo conforme a fines educativos, orientación o adopción.
Sector objetivo: En qué sector pretende tener impacto, académico o industrial.
Apego a la ISO 42010: Si se cuenta con algún apego a la ISO de definición de arquitecturas de referencia 42010.
Apego Norma IEEE 2675: Si se cuenta con las bases de la norma internacional del IEEE 2675.
Apego a herramientas: Si la arquitectura de referencia está ligada a utilizar herramientas de software específicas para trabajar con ella. En tal caso, es necesario tener un apego a las herramientas disponibles y utilizarlas de manera efectiva para lograr sus objetivos con la arquitectura, lo que le resta flexibilidad a una arquitectura de referencia [13].
Marcos de referencia arquitectura de Referencia: Sí se desarrolló la arquitectura bajo algún marco. Los marcos proporcionan una guía para la creación, mejorar la comprensión de lo que son las arquitecturas de referencia, así como sus componentes y relaciones, el establecimiento, uso y evolución de dichas arquitecturas [24].
Uso libre: En el contexto de una arquitectura de referencia se refiere a las restricciones o permisos en cuanto al uso de los componentes de la arquitectura. En algunos casos, los componentes pueden ser de uso libre, lo que significa que pueden ser utilizados sin

restricciones. En otros casos, pueden estar sujetos a ciertas limitaciones, como licencias de software o acuerdos de uso específicos.

Tabla 2. Comparativa de trabajos relacionados

Arquitecturas de Referencia académicas	Objetivo	Sector Objetivo	Apego a la ISO 42010	Apego Norma IEEE 2675	Apego a herramientas	Marcos de referencia para la arquitectura	Uso libre
Arquitectura de referencia de IBM [31].	Adopción en la industria.	Académico/ industrial	Sí	No	No	Sí	Sí
DevOps Microsoft, Azure DevOps [39].	Adopción en la industria.	industrial	Sí	No	Sí	Sí	Limitado
DevOps AWS [40].	Adopción en la industria.	industrial	Sí	No	Sí	Sí	Limitado
A reference architecture for the container ecosystem [41].	Adopción en la industria para el ecosistema de los contenedores.	Industrial	No	No	No	No	Sí
DevOps Reference Architecture for Multi-Cloud IOT Applications [42].	Orientación en la implementación sistemática de DevOps para el despliegue de IoT en multicloud.	Académico	No	No	No	No	Sí
A Contribution to the Establishment of Reference Architectures for Mobile Learning Environments [43].	Orientación para desarrollo en entornos de aprendizaje móviles.	Académico	No	No	No	No	Sí
CALMS [44].	Adopción en la industria Pymes.	Industrial	No	No	No	Sí	Limitado
Three ways [45]	Adopción en la industria.	industrial	No	No	No	No	Sí
ARA (Arquitectura de referencia DevOps Académica).	Enseñanza	Académico	Sí	Sí	No	Sí	Sí

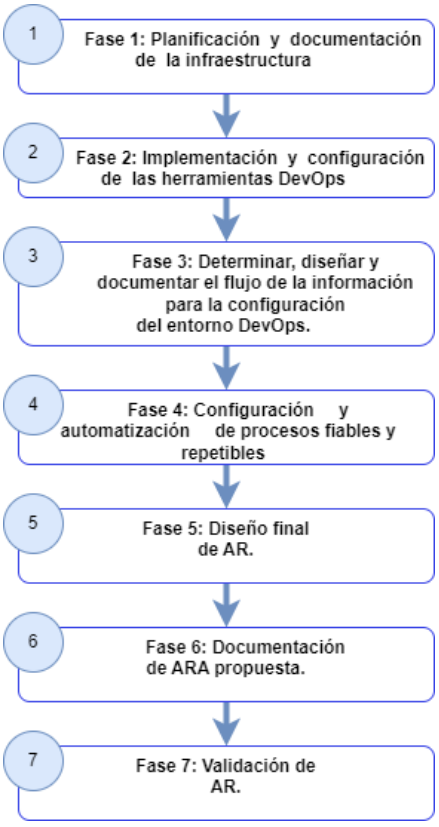
Capítulo IV

4. Metodología de solución

En este capítulo, se presenta una metodología de solución para el diseño y desarrollo de una arquitectura de referencia académica de DevOps. La metodología se ha diseñado específicamente para abordar los desafíos y alcanzar los objetivos establecidos en el desarrollo de proyectos de software en entornos DevOps. Además, su objetivo principal es facilitar la integración y orientación en la práctica de los fundamentos de DevOps.

Más adelante se describe en detalle cada uno de los pasos de la metodología de solución, incluyendo las herramientas y técnicas utilizadas en cada fase del proceso. Al seguir esta metodología, los estudiantes de las áreas de Ingeniería de Software pueden beneficiarse de este enfoque al seguir paso a paso la arquitectura de referencia (AR) propuesta para obtener competencias específicas en el desarrollo de software en entornos DevOps.

La Fig. 1 muestra las fases de la metodología del desarrollo de ARA basadas en la metodología RAModel [24] y en estándares específicos de DevOps, así como herramientas y fuentes importantes en el desarrollo de esta investigación.



AR: Arquitectura de Referencia

Fig. 1. Fases de desarrollo.

4.1. Fase 1: Planificación y documentación de la infraestructura.

En esta fase se establecen los planes y se crea la documentación necesaria para la infraestructura que se utilizó en la creación de la arquitectura de referencia. Esta fase implica la definición detallada de los recursos, configuraciones y arquitectura, necesarios para respaldar el desarrollo, la implementación y el funcionamiento de la arquitectura de referencias propuesta. Esta fase es crucial para garantizar que todos los aspectos de la infraestructura estén bien definidos y comprendidos antes de avanzar en las etapas de implementación y operación.

Los siguientes aspectos fueron analizados para la planificación y la configuración de la infraestructura de la arquitectura de referencia propuesta, las Tablas 3 y 4 muestran la clasificación de dichos aspectos relacionados con las referencias normativas y la literatura de referencia analizada para el desarrollo y diseño de la arquitectura de referencia académica (ARA).

1. Análisis y comprensión del enfoque DevOps
-Mejores prácticas
-Actualidad y popularidad
-Información general
-Taxonomía de DevOps
-Estándares (IEEE 2675:2021)
-Metodologías ágiles
2. Estudio y análisis de modelos y arquitecturas de referencia
-Tipos de modelos y arquitecturas de referencia
- Análisis comparativo de AR académicas vs. industriales
-Arquitecturas existentes de DevOps
-Metodologías de desarrollo de AR
-Metodologías de validación de AR
-Tipos de vistas dentro de AR
-Herramientas para diseño de Arquitecturas
-Modelos y decisiones arquitectónicas
3. Academia
-DevOps y su enseñanza
-Habilidades requeridas para DevOps
-Métodos de enseñanza de DevOps
-Selección de taxonomía académica

Tabla 3. Referencias normativas relacionadas

Nombre	Desarrollado por	Año de Publicación	Descripción
Estándar IEEE 2675:2021	Comité de Normas de ingeniería de software y sistemas de la sociedad de informática del IEEE.	2021-02-09	El objetivo de esta norma es especificar las prácticas necesarias para que las operaciones, el desarrollo y otras partes interesadas clave colaboren y se comuniquen para desplegar sistemas y aplicaciones de forma segura y fiable.
Norma 42010	Comité de Normas de ingeniería de software y sistemas de la sociedad de informática del IEEE.	2011-12-01	Esta Norma Internacional aborda la creación, el análisis y el mantenimiento de arquitecturas de sistemas mediante el uso de descripciones de estas.
Reference Architecture Model (RAModel)	Nakagawa, E. Y., Oquendo, F. and Becker, M, Proceedings of the 2012 Joint Working Conference on Software Architecture and 6th European Conference on Software Architecture, WICSA/ECSA 2012, pp. 297–301. doi: 10.1109/WICSA-ECSA.212.49.	2012	Modelo de referencia para arquitecturas de referencia, que pretende mejorar la comprensión de lo que son las arquitecturas de referencia, así como sus componentes y relaciones.

Tabla 4. Literatura de referencia relacionada

Nombre	Autores	Año de publicación	Editorial
Continuous Delivery [1]	Jez Humble and David Farley	2010	Addison-wesley
DevOps para Dummies [31]	Sanjeev Sharma Bernie Coyne	2015	WILEY
Engineering DevOps [27]	Marc Hornbeek	2019	a.k.a. DevOps_The_Gray esq.
Continuous Integration Improving software quality and reducing risk [46]	Paul M. Duvall with Steve Matyas and Andrew Glover	2007	Addison-wesley
The DevOps Handbook [47]	Gene Kim, Jez Humble, Patrick Debois, and John Willis	2016	IT Revolution
Learning DevOps [48]	Mikael Krief	2019	Packt

Llevar a cabo esta fase fue una parte muy importante del trabajo, ya que permitió la definición de los procesos clave del paradigma DevOps, que más adelante serán conocidos en este proyecto como categorías de software definidas por las etapas de la metodología DevOps.

Estos procesos clave identificados (categorías) son:

- Planificación y seguimiento
- Control de versiones
- Integración continua
- Despliegue continuo
- Implementación
- Pruebas
- Monitoreo y registro
- Gestión de la configuración

4.2. Fase 2: Implementación y configuración de las herramientas DevOps.

Durante esta fase, y con respecto a la fase anterior, se examinó la aplicación de las prácticas y procesos de DevOps, así como los desafíos y beneficios que se han obtenido en diferentes contextos (Anexo G). Esto ayudó a comprender mejor las implicaciones prácticas de DevOps. Para practicar la aplicación de estos principios, se desarrolló una taxonomía que, a su vez, ayudó a configurar un servidor local para la práctica posterior. Algunos de los pasos dentro de esta fase fueron:

Paso 1. Definir una taxonomía a partir de herramientas DevOps

Se generó una taxonomía, la cual, según [49], [50] un sistema de categorización y clasificación que permite organizar y ordenar información de manera eficiente, de tal manera que se pueda acceder y utilizar más fácilmente. Una taxonomía de herramientas DevOps puede ser útil por tres razones:

- Organización y clasificación, una taxonomía puede ayudar a organizar y clasificar las diferentes herramientas disponibles en el mercado para las diferentes etapas del ciclo de vida de desarrollo de software. Esto puede facilitar la selección de herramientas apropiadas para un proyecto en particular y reducir el tiempo y la energía que se necesitan para investigar y explorar diferentes opciones [49], [50].
- Una comprensión compartida de las diferentes herramientas, sus funciones, y cómo se comunican entre sí. Esto permite una comunicación más efectiva y la toma de decisiones informadas.
- Compatibilidad: una taxonomía, también puede ayudar a garantizar la compatibilidad y la integración de las diferentes herramientas que se utilizan en el ciclo DevOps. Al clasificar y organizar las herramientas por función y etapa, se pueden identificar las mejores opciones para cada tarea y asegurarse de que sean compatibles y se integren bien en el proceso de desarrollo.
- Debido a la necesidad de tener una clasificación con características especiales afines a la academia, se decidió definir una taxonomía que permitiera seleccionar las mejores herramientas para implementar un escenario DevOps.

Paso 2. Características del software a ser consideradas.

Categorías de software definidas por las etapas de la metodología DevOps que fueron consideradas en esta taxonomía: planificación y seguimiento, control de versiones, integración continua, despliegue continuo, implementación, pruebas, monitoreo y gestión de la configuración. Existen diversas características que un software debe tener para ser considerado adecuado para su uso en entornos educativos, especialmente en escuelas.

Características generales, tomadas en cuenta para esta investigación: Software libre o tener una versión gratuita para practicar, Popularidad en la industria, Interfaz de usuario intuitiva, seguridad y privacidad, flexibilidad, escalabilidad, contenido educativo de calidad, Interactividad, retroalimentación, personalización e integración.

Todas las características se incluyen en el **Anexo A.**

La Tabla 5 presenta un resumen de las herramientas que se han revisado para el desarrollo de la taxonomía de este proyecto, así como nuestras sugerencias de las herramientas en cuanto a su participación en el ciclo DevOps:

Tabla 5. Herramientas revisadas para el desarrollo de la taxonomía.

Etapa de DevOps	Herramientas de software libre	Herramientas sugeridas
Planificación y seguimiento	GitLab [51], GitHub [52], Jira [53], Trello [54], Asana [55], Redmine [56], Taiga [57], Kanboard [58], Wekan [59]	GitLab
Control de versiones	Git [60], Bitbucket [61], SVN [62], SourceForge [63], Mercurial [64]	GitLab
Integración continua	Jenkins [65], Travis CI [66], CircleCI [67], Codeship [68], semaphore [69]	Jenkins
Implementación	Ansible [70], Puppet [71], Chef [72], Terraform [73], SaltStack [74]	Ansible
Pruebas	Selenium [75], JUnit [76], TestNG [77], Cucumber[78], Robot Framework [79], Appium [80], Cypress [81], SoapUI [82]	Selenium
Monitoreo y registro	Prometheus [83], Grafana[84], Nagios [85], Zabbix [86], Icinga [87], Cacti [88], Sensu [89], Monit [90], Datadog [91], Splunk [92]	Prometheus, Grafana
Gestión de la configuración	Kubernetes [93], Terraform [73], Docker Compose [94], Vagrant [95], OpenStack [96]	Docker Compose
Despliegue continuo	TeamCity [97], Bamboo [98], Concourse CI [99], GoCD [100], Jenkins [65], Travis CI [66], GitLab CI/CD [51], CircleCI [67], TeamCity [97], Codeship [68]	Jenkins, GitLab CI/CD

La Tabla 6 muestra las herramientas que se seleccionaron para instalar en un servidor local y construir un entorno de pruebas con herramientas DevOps.

Tabla 6. Selección de herramientas para desarrollar un entorno DevOps de prueba

Herramienta	Etapas del ciclo DevOps en las que se utiliza	Sugerencia
Jenkins [65]	Construcción e integración continua	Recomendada para equipos que buscan una herramienta de automatización de CI/CD estable y con gran cantidad de plugins disponibles.
Docker [94]	Despliegue y contenerización	Altamente recomendada para la creación y despliegue de aplicaciones en contenedores, lo que permite la portabilidad y escalabilidad de las mismas.

Herramienta Etapas del ciclo DevOps en las que se utiliza Sugerencia	Herramienta Etapas del ciclo DevOps en las que se utiliza Sugerencia	Herramienta Etapas del ciclo DevOps en las que se utiliza Sugerencia
Ansible [70]	Orquestación y configuración	Recomendada para equipos que buscan una herramienta de automatización de configuración y orquestación de infraestructura simple y fácil de usar.
Kubernetes [93]	Orquestación de contenedores y gestión de aplicaciones	Altamente recomendada para el despliegue y gestión de aplicaciones en contenedores, lo que permite la escalabilidad y alta disponibilidad de las mismas.
GitLab [51]	Control de versiones, CI/CD, despliegue y monitoreo	Altamente recomendada para equipos que buscan una plataforma integral que les permita gestionar todo el ciclo de vida de software de manera efectiva.
Prometheus [83]+ Grafana [84]	Monitoreo y registro	Juntas, estas herramientas proporcionan una solución completa para la monitorización y visualización de los sistemas en el contexto de DevOps. Permiten a los equipos de desarrollo y operaciones monitorear y analizar el rendimiento de sus sistemas, identificar de manera temprana, problemas de rendimiento, lo que ayuda a mejorar la calidad del software y la eficiencia del ciclo de vida de desarrollo.
Selenium [75]	Pruebas	Una de las ventajas de Selenium es que puede integrarse fácilmente en sistemas de automatización de CI/CD para automatizar el proceso de pruebas. Esto significa que las pruebas pueden ejecutarse automáticamente cada vez que se realiza una nueva actualización de código, lo que permite una detección temprana de errores y reduce el tiempo y esfuerzo necesarios para realizar pruebas manuales.

Algo importante que mencionar al realizar la revisión de una gran variedad de software en esta etapa, es que GitLab es una plataforma integral que puede participar en varias etapas del ciclo DevOps, lo que la convierte en una herramienta valiosa para el desarrollo de proyectos en la enseñanza de la Ingeniería de Software. En general, GitLab es una herramienta altamente recomendada para el desarrollo de proyectos, especialmente para aquellos equipos que buscan una solución completa y flexible para su ciclo de vida de software. Sin embargo, existen más herramientas que pueden ser una mejor opción según sea el caso.

En resumen, la selección de herramientas dependerá del contexto y las necesidades específicas de cada equipo de desarrollo. Es importante evaluar cuidadosamente las opciones disponibles y elegir las herramientas que mejor se ajusten al proyecto en cuestión. Sin embargo, para fines de práctica y realización de este proyecto, se seleccionaron las herramientas que se muestran en la Tabla 6.

Paso 3. Definir el tipo de esquema a diseñar
Como paso siguiente al desarrollo de la taxonomía, se definió el tipo de esquema a diseñar. Debido a los objetivos de este proyecto, se seleccionó un esquema conceptual para identificar los principios y procesos clave del paradigma DevOps y, así, integrarlos en una estructura básica.

La vista conceptual, también conocida como esquema conceptual, se utiliza a menudo como punto de partida para el desarrollo de otras vistas más detalladas, como la vista lógica y la vista física. Esta vista es importante porque proporciona una comprensión común del sistema en todas las fases del ciclo de vida de un software. Además, sirve como guía para el diseño y construcción de sistemas de manera coherente y consistente. La vista conceptual se compone de conceptos y teorías que relacionan su funcionalidad y comunicación, y que utiliza algunos símbolos para orientar al lector, sin incluir imágenes [17], [18], [22], [25].

Para el desarrollo de la vista conceptual, se siguió la metodología de RAModel, la cual propone pautas para el desarrollo de arquitecturas de referencia. Esta metodología nos sirvió como orientación en el desarrollo de las versiones de la arquitectura de este proyecto.

Paso 4. Toma de decisiones arquitectónicas.
En este paso se creó una taxonomía de herramientas DevOps, dividida en nueve categorías previamente definidas: planificación y seguimiento, control de versiones, integración continua, despliegue continuo, implementación, pruebas, monitoreo y gestión de la configuración. Además, se tomaron decisiones importantes sobre qué tipo de vista era la más adecuada para la arquitectura de referencia y cómo se desarrollaría la arquitectura siguiendo la metodología RaModel.

4.3. Fase 3: Determinar, diseñar y documentar el flujo de la información para la configuración del entorno DevOps.

Durante esta fase, se llevó a cabo la identificación, diseño y documentación del proceso para establecer el entorno DevOps. El objetivo fue determinar el flujo de la información necesario para su configuración.

1. **Instalación y configuración de herramientas en un servidor:**

En esta etapa, se procedió a la instalación y configuración de las herramientas necesarias en un servidor, además de documentar las instalaciones, ver **ANEXO B**. Se identificaron las herramientas requeridas según la taxonomía propuesta y se siguieron

los pasos correspondientes para su instalación. Esto implicó la descarga de los paquetes de instalación, su ejecución y la configuración de los parámetros necesarios.
Se verificó que los requisitos del sistema fueran cumplidos, como la compatibilidad de las herramientas con el sistema operativo y los recursos de hardware necesarios. Se siguieron las instrucciones proporcionadas por los proveedores de las herramientas para asegurar una instalación correcta.

Además, se realizó la configuración de las herramientas de acuerdo a los requisitos del proyecto y la taxonomía propuesta. Esto incluyó la definición de usuarios y permisos, la configuración de conexiones a bases de datos o servicios externos, la personalización de las opciones de configuración y cualquier otro ajuste necesario para adaptar las herramientas a las necesidades del proyecto.

2. Configuración de las herramientas de la taxonomía propuesta:
Una vez instaladas las herramientas en el servidor, se procedió a su configuración de acuerdo a la taxonomía propuesta. Esto implicó establecer los parámetros y opciones de configuración específicos requeridos por la taxonomía, asegurando la consistencia y coherencia en el uso de las herramientas.
Se definieron las estructuras de directorios, los nombres de archivos y cualquier otro elemento necesario para cumplir con la taxonomía propuesta. También se establecieron los flujos de trabajo y las reglas de negocio relacionadas con las herramientas, asegurando su correcto funcionamiento dentro del contexto de la taxonomía.

Además, se realizaron pruebas de funcionalidad y se verificó que las herramientas configuradas se integrarán adecuadamente entre sí y con los componentes del sistema. Se corrigieron posibles problemas o conflictos que surgieran durante esta etapa de configuración (Ver **anexo C**).

En resumen, se llevó a cabo la instalación y configuración de las herramientas en el servidor, siguiendo los pasos necesarios y atendiendo los requisitos de compatibilidad y personalización. Se estableció la configuración de acuerdo a la taxonomía propuesta, asegurando la coherencia y consistencia en el uso de las herramientas en el proyecto.

Como resultado de esta etapa, se desarrolló el esquema de la Fig.2 que muestra las herramientas seleccionadas para construir un entorno DevOps local con fines de práctica.

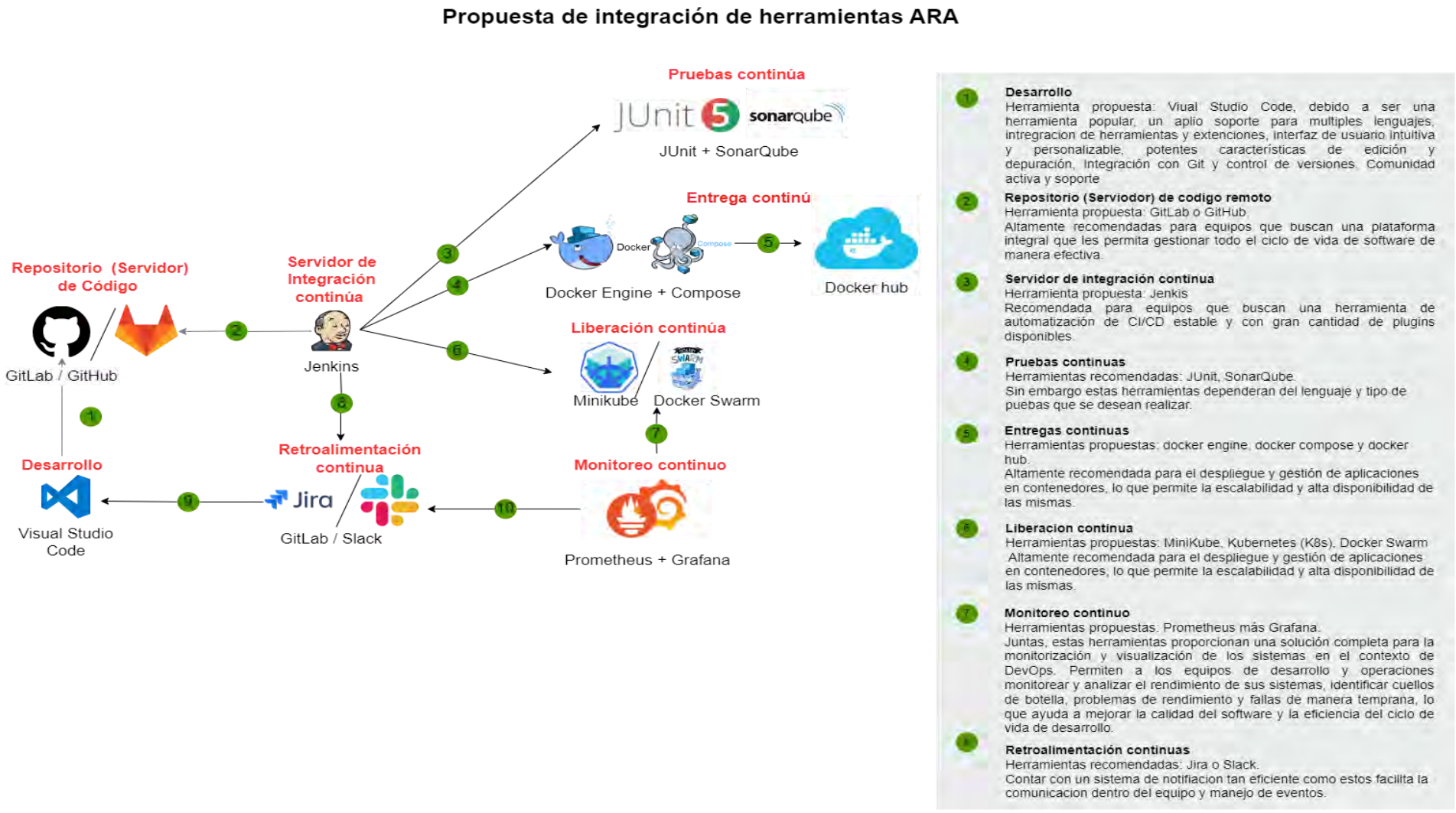


Fig. 2. Esquema de herramientas para configurar un entorno DevOps.

4.4. Fase 4: Configuración y automatización de procesos DevOps.

Durante esta fase, se llevó a cabo la configuración, diseño y planificación de pipeline y automatización de procesos DevOps, basados en los siguientes procesos:

1. Diseño y planificación de pipeline (propuesta) con base a estándar DevOps:

En esta fase, se diseñó y planificó el pipeline de entrega continua (Continuous Delivery) siguiendo los estándares y prácticas de DevOps (Ver Configuración de procesos DevOps en **ANEXO C**). Se identificaron las fases del pipeline, como compilación, pruebas unitarias, pruebas de integración, pruebas de rendimiento, empaquetado, despliegue y monitoreo. Se definieron los requisitos y criterios de calidad para avanzar en el pipeline, estableciendo pruebas automatizadas, estándares de codificación, métricas de rendimiento y requisitos de seguridad. Se seleccionaron y configuraron las herramientas necesarias, como sistemas de control de versiones, sistemas de compilación, herramientas de pruebas y orquestadores de despliegue. Se automatizó cada etapa del pipeline utilizando herramientas y scripts, asegurando un proceso de entrega consistente y reproducible. Además, se estableció una estrategia de rollback para gestionar posibles problemas en el despliegue y permitir una rápida reversión a una versión anterior funcional en caso de fallos.

2. Desarrollo de flujo de trabajo como guía para la elaboración de la Arquitectura de Referencia (AR):

Una vez que se diseñó el pipeline, se utilizó como guía para la elaboración de la Arquitectura de Referencia (AR). El flujo de trabajo del pipeline proporcionó una estructura clara y orientación sobre cómo los diferentes componentes del sistema interactúan y se integran en el contexto de DevOps. Se identificaron los componentes clave del sistema, como servicios, aplicaciones, bases de datos y sistemas de almacenamiento. Se definieron las interfaces y las dependencias entre los componentes, comprendiendo cómo se comunican y colaboran entre sí. Se establecieron las mejores prácticas de diseño arquitectónico, asegurando la modularidad, escalabilidad, tolerancia a fallos, seguridad y otros principios de diseño clave. Asimismo, se documentaron los patrones de diseño utilizados en la AR, incluyendo patrones de arquitectura, patrones de integración, patrones de despliegue y otros patrones relacionados con DevOps.

4.5. Fase 5: Diseño final de AR.

Durante esta fase, se diseñó la arquitectura basándose en los siguientes procesos:

- Diseñar la arquitectura de referencia, incluyendo la definición de los componentes a partir de la práctica e investigación y bajo la toma de decisiones arquitectónicas elegidas.
- Revisión de alcances, objetivos y limitaciones de la AR.

- Revisión de la versión preliminar de la AR con las decisiones arquitectónicas tomadas.
- Personalización de la AR (Nombre elegido ARA).
  Como resultado de la personalización de ARA se desarrolló su logotipo, ver Fig.3.

Fig. 3. Logotipo de ARA.

4.6. Fase 6: Documentación de ARA propuesta.

Durante esta etapa, se documentó la propuesta de ARA basándose en los siguientes procesos:

- Crear la estructura o plantilla de la guía de ARA.
- Desarrollo de documentación para la arquitectura ver **Anexo D**.

4.7. Fase 7: Validación de AR.

La validación de una AR puede ser cualitativa o cuantitativa. La validación cuantitativa se puede realizar mediante un caso de uso, mientras que la validación cualitativa se realiza a través del juicio de expertos [43].

Los procesos dentro de esta etapa de validación fueron los siguientes:

- Selección del proceso del método de validación por "Validación de ARA mediante la técnica juicio de expertos (EJT, Expert Judgment Techniques) de la industria y académicos e investigadores en DevOps."
- Planificación de entrevistas ver **Anexo E**.
- Entrevistas con profesionales **Anexo F**.
- Análisis de resultados (**capítulo V**).

En la etapa de validación de la arquitectura de referencia (AR), se utilizó la técnica de juicio de expertos/profesionales [101] y [102] para una validación cualitativa, también conocido como método Delphi [103]. Cabe destacar que se usó este método de validación, por su reconocimiento y uso en otras arquitecturas y modelos de referencia, tal como se menciona en [43], [103] y modelos de DevOps como en [104].

Juicio de Expertos / Expert Judgement [Técnica]. Según el cuerpo de conocimientos de PMBOK Cuarta edición, "un juicio que se brinda sobre la base de la experiencia en un área de aplicación, área de conocimiento, disciplina, industria, etc. según resulte apropiado para la actividad que se está llevando a cabo. Dicha experiencia puede ser proporcionada por cualquier grupo o persona con una educación, conocimiento, habilidad, experiencia o capacitación especializada" [101].

Según [103], *"los expertos del dominio emiten juicios y opiniones compartidas basadas en su experiencia, habilidad o conocimientos en un área concreta. Los expertos pueden ser un jefe de equipo, un gestor de proyectos, partes interesadas, consultores y expertos en la materia o el dominio"*.

Según PMBOK un Experto se define como: "Todo grupo o individuo con conocimientos o capacitación especializados y se encuentra disponible a través de diferentes fuentes, entre las que se incluyen: consultores, interesados, incluyendo clientes y patrocinadores, asociaciones profesionales y técnicas, grupos industriales, expertos en la materia y la oficina de dirección de proyectos" [101].

En lo referente al número máximo o mínimo de expertos para el uso del método Delphi, según [105] *"no existe un consenso en la literatura sobre el número óptimo de expertos en un desarrollo Delphi, a pesar de que este paso se muestra crítico en el desarrollo del método".*

Capítulo V

5. Resultados y validación

En este capítulo, se presentarán los resultados y la validación de la Arquitectura de referencia académica de DevOps propuesta en el capítulo anterior.

5.1. Diseño de la Arquitectura de Referencia Académica de DevOps (ARA)

La Arquitectura de referencia académica de DevOps propuesta en este trabajo se nombró como ARA y se plantea a partir de dos capas de abstracción modular, como se muestra en la Fig. 4. El módulo, "A", abarca los “Procesos de gestión y configuración de infraestructura", este comienza por establecer y configurar un entorno de trabajo a partir de servicios básicos y comunes que se establecen para poder practicar con DevOps. El módulo, "B", busca poner en práctica los “Procesos del ciclo de vida DevOps y las entradas/salidas” que se obtienen a partir de la ejecución de estas prácticas. La información detallada de estos módulos se encuentra disponible en el **ANEXO D**.

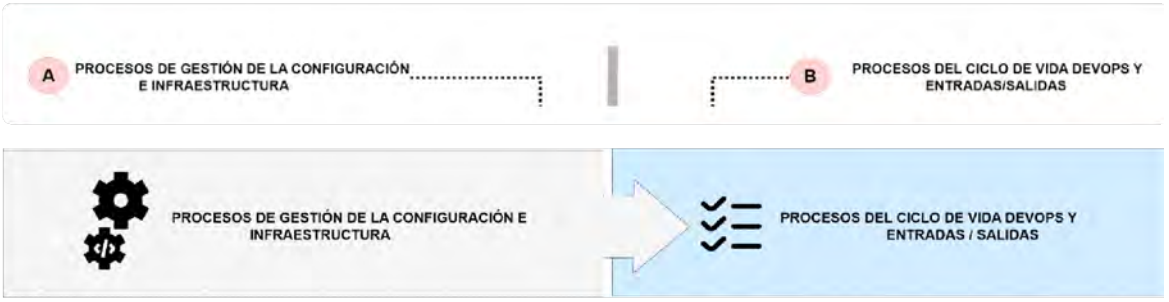


Fig. 4. Primera capa de abstracción modular de ARA.

Módulo A.

Procesos de gestión y configuración de infraestructura

El módulo “A” está enfocado en la gestión y configuración de la infraestructura necesaria para la práctica de DevOps. En este módulo se establece y configura un entorno de trabajo que incluye servicios básicos y comunes, con el objetivo de permitir la implementación de DevOps de manera práctica. Cuenta con una sub-modularización que comparte con la sección “B” y en cada módulo se detalla la gestión de estos seis niveles o módulos (Ver Fig. 5).

Módulo B.

El módulo "B" tiene como objetivo poner en práctica los procesos del ciclo de vida DevOps, incluyendo las entradas y salidas que se obtienen a partir de la ejecución de estas prácticas.

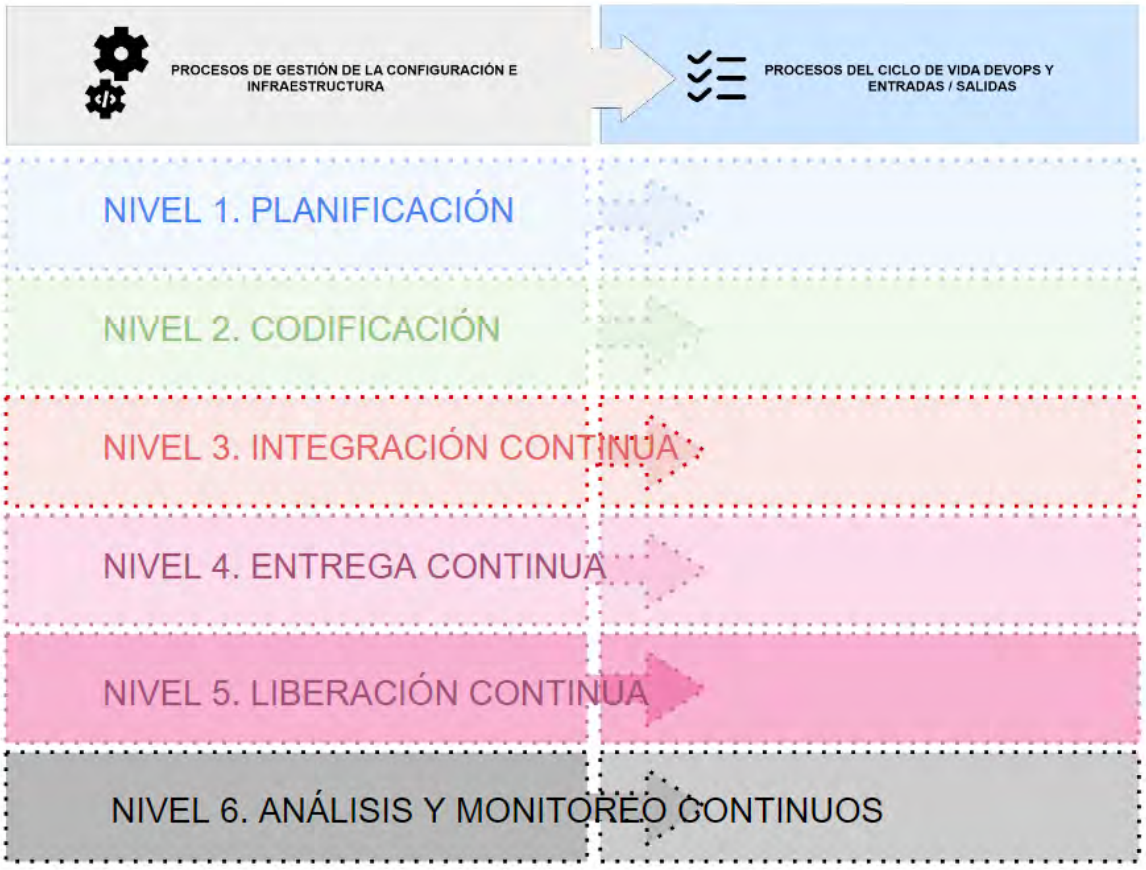


Fig. 5. Segunda capa de abstracción modular de la ARA.

Después de ocho versiones propuestas, se llegó a la propuesta de la versión final de la Arquitectura de Referencia Académica de DevOps (ARA) como se puede ver en la Fig. 6 y que se generó a partir de la metodología antes descrita.

5.2. Arquitectura de referencia académica de DevOps (ARA)

La Arquitectura de referencia académica de DevOps pretende ayudar a los equipos de desarrollo a implementar DevOps en su proceso de desarrollo de software. La Fig. 6 proporciona una vista general de esta metodología y sus componentes clave.

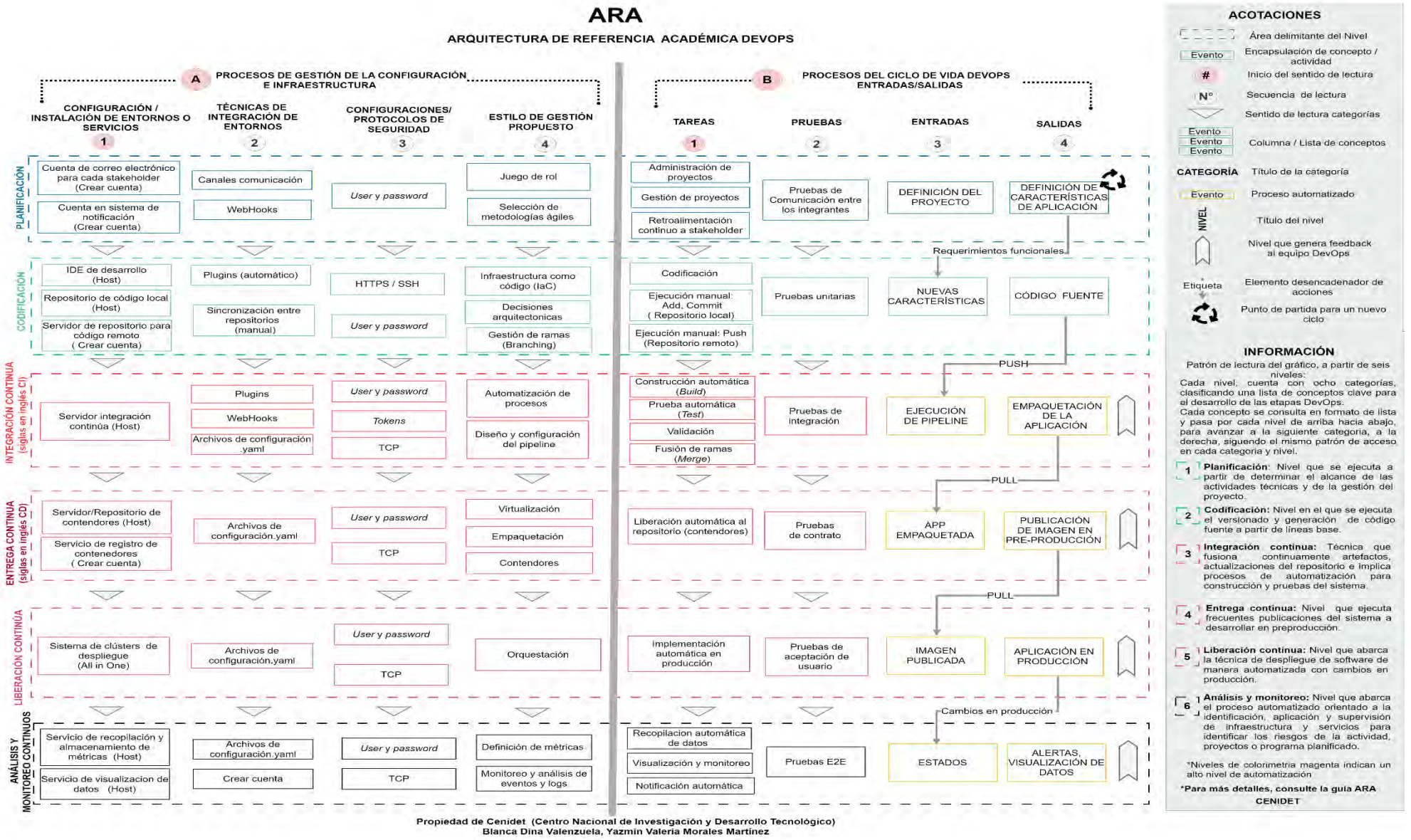
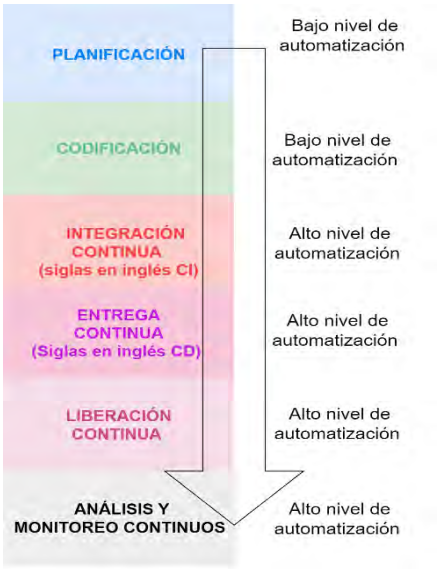


Fig. 6. Arquitectura de Referencia Académica DevOps.



Niveles de automatización de la arquitectura ARA

Cada nivel cuenta con un nivel de automatización, como se ve en la Fig. 7, que va del nivel más bajo de automatización en la parte superior y de colores magenta o rojos, los niveles que involucran una alta automatización, por último, el nivel de color gris contiene partes de automatización, y algunas otras partes como el análisis y atención a eventos de manera manual.

Fig. 7. Niveles de automatización de la arquitectura ARA.

Categorías en los niveles

Cada nivel, cuenta con ocho categorías, y una lista de eventos clave que proporcionan una estructura sólida y organizada para aplicar los principios y prácticas de DevOps (ver Fig.8). Inicialmente, los eventos se consultan en formato de lista y continúa por cada nivel de arriba hacia abajo, al finalizar se continúa con la siguiente categoría que se encuentran a la derecha, siguiendo el mismo patrón de acceso en cada nivel.

Categorías dentro del primer módulo

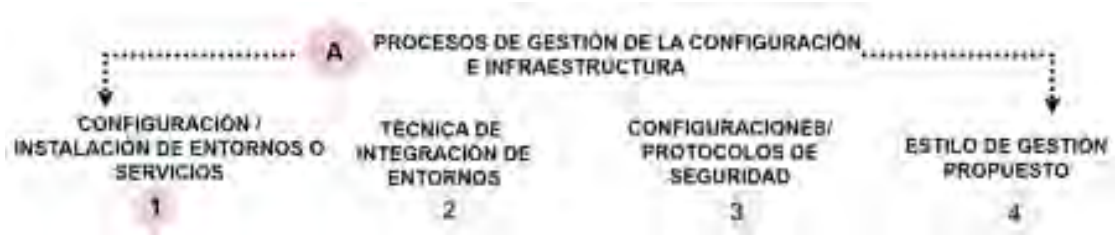


Fig. 8. Categorías del módulo "A" de ARA.

Categoría 1) **"Configuración/instalación de entornos o servicios"** (ver Fig.10)**.** Abarca los eventos o tareas que se llevan a cabo a partir de hacer unas configuraciones e instalaciones o montar servicios básicos que se deben considerar para comenzar a trabajar bajo las prácticas de DevOps, los términos encapsulados en recuadros, cuenta con palabras clave que indican el tipo de instalación entre paréntesis, como:

- Host: que se refiere a instalar o configurar un programa, aplicación o software en la máquina o equipo principal en el que se está trabajando. El "Host" en este contexto hace referencia al sistema operativo o la infraestructura principal en la que se ejecuta el software.

- Crear cuenta: implica la creación de una cuenta con algún proveedor de algún servicio o software clave en la participación en el entorno DevOps.
- Instalación “All in One”: Tipo de instalación, en el que se instala y configura un sistema o software completo en un único dispositivo o servidor. En lugar de distribuir los componentes del sistema en diferentes servidores o máquinas virtuales, todos los componentes se instalan y ejecutan en una sola instancia.

Categoría 2) “Técnicas de integración de entornos”. Las técnicas de integración de entornos son procesos y prácticas utilizadas para asegurar que los diferentes componentes de un sistema de software funcionen juntos de manera efectiva.

Categoría 3) Esta categoría presenta las configuraciones y/o protocolos básicos para establecer la seguridad, no se consideran técnicas de seguridad más profundas debido a las intenciones limitadas de esta Arquitectura, sin embargo, para profundizar en el tema de seguridad, considere el uso de DevSecOps.

Categoría 4) A partir de esta categoría y después de haber configurado e integrado un entorno de trabajo DevOps básico, comenzaremos a trabajar bajo los estilos propuestos de ARA. Estos estilos incluyen técnicas y procesos que involucran la simulación para ejercitar las prácticas DevOps. Cabe mencionar que estas técnicas son de uso frecuente en la industria y fueron seleccionadas a partir de estudios, investigaciones y entrevistas a expertos que se desempeñan en el campo industrial del desarrollo de software.

Categorías dentro del segundo módulo

Categoría 1) lista las tareas y prácticas más importantes para comenzar a trabajar bajo un enfoque DevOps (ver Fig. 9).

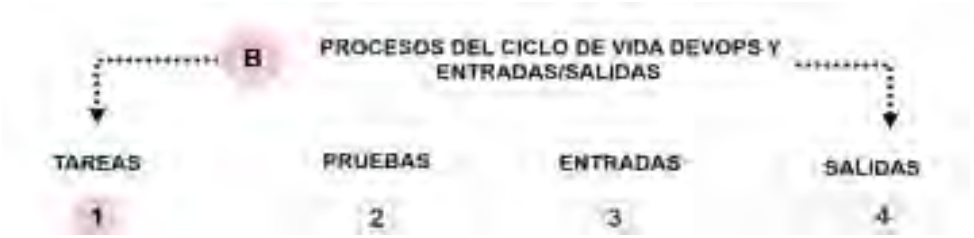


Fig. 9. Categorías del módulo “B” de ARA.

Categoría 2) Esta segunda categoría del módulo de procesos**,** lista algunos de los tipos de pruebas presentes en las diferentes etapas del ciclo de vida de DevOps que son parte de un ciclo de prueba continúa.

Categorías 3 y 4) Estas categorías presentan las entradas y salidas en cada nivel, a partir de la ejecución de tareas y procesos anteriormente ejecutadas.

Conceptos dentro de cada categoría y nivel

En la siguiente sección se muestran los procesos de ejecución de ARA, estos procesos son independientes al uso y selección de herramientas y conceptos dentro de ellas, la selección de herramientas queda a criterio libre e independiente de cada usuario que pretenda trabajar bajo este esquema, es importante mencionar que en la elección de las herramientas debe considerarse a partir de los aspectos particulares de cada proyecto, uso de licencias libres o de paga y propósitos específicos.

Estos son solo los pasos generales, los pasos específicos dependen fuertemente del software elegido y la configuración puede variar. Asegurarse de elegir las mejores herramientas que se adapten a sus proyectos o investigaciones.

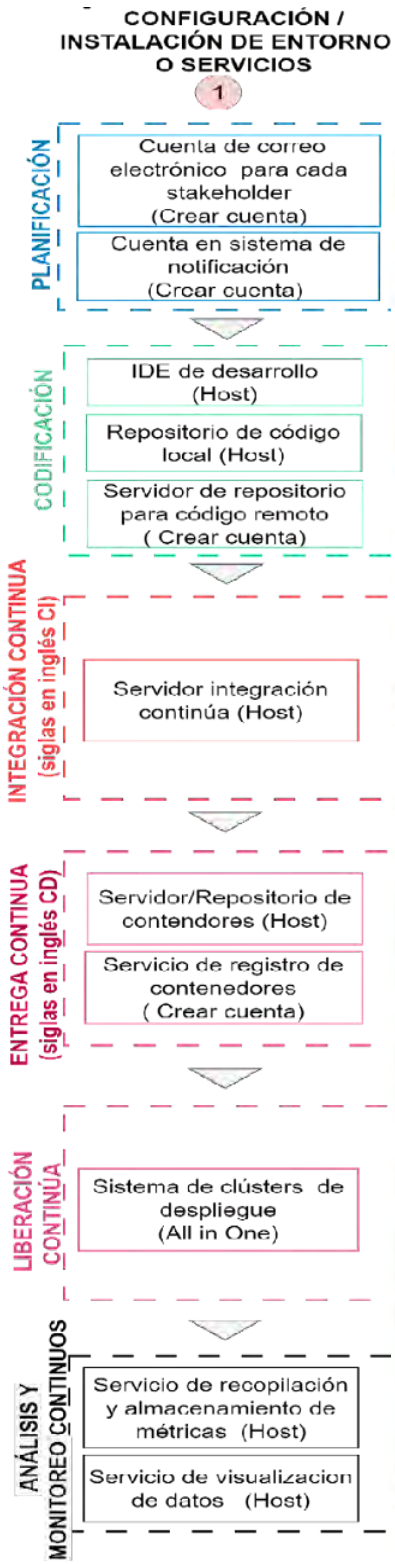
A continuación, se presenta una descripción de los eventos y conceptos contenidos en los recuadros. En el lado izquierdo se encuentra una imagen que muestra la ubicación secuencial de estos elementos, mientras que en el lado derecho se ofrece un desglose detallado de cada uno de los conceptos. Esta presentación visual y descriptiva permite una comprensión clara y organizada de los contenidos, facilitando su seguimiento y asimilación.

5.2.1. Conceptos dentro de cada categoría y nivel

A continuación, se muestran los procesos de ejecución de ARA, estos procesos son independientes al uso y selección de herramientas y conceptos dentro de ellas, la selección de herramientas queda a criterio libre e independiente de cada usuario que pretenda trabajar bajo este esquema, es importante mencionar que en la elección de las herramientas debe considerarse a partir de los aspectos particulares de cada proyecto, uso de licencias libres o de paga y propósitos específicos.

Estos son solo los pasos generales, los pasos específicos dependen fuertemente del software elegido y la configuración puede variar. Asegurarse de elegir las mejores herramientas que se adapten a sus proyectos o investigaciones.

A continuación, se presenta una descripción de los eventos y conceptos contenidos en los recuadros. En el lado izquierdo se encuentra una imagen que muestra la ubicación secuencial de estos elementos, mientras que en el lado derecho se ofrece un desglose detallado de cada uno de los conceptos. Esta presentación visual y descriptiva permite una comprensión clara y organizada de los contenidos, facilitando su seguimiento y asimilación.



Conceptos dentro de cada categoría y nivel

PLANIFICACIÓN

Cuentas de correo electrónico: Se sugiere crear cuentas de correo electrónico personales para cada interesado, por ejemplo, Gmail o Outlook, entre otros.

Sistema de notificación

Se debe seleccionar el sistema de notificación que mejor se adapte a las necesidades y requerimientos del proyecto. Se pueden utilizar sistemas existentes como Jira o Slack.

CODIFICACIÓN

Seleccionar e instalar el IDE de desarrollo: Para seleccionar un IDE es recomendable buscar uno que tenga una amplia posibilidad de conexión con otros entornos y que se adapte a las necesidades y gustos del usuario. Algunas opciones de IDEs que se pueden considerar son Visual Studio Code o IntelliJ.

Configurar un repositorio de código local: Crear un repositorio en la computadora local en donde se pueda almacenar y gestionar el código fuente de un proyecto de software.

Servidor de repositorio para código remoto: Un servidor de repositorio para código remoto se refiere a una plataforma o servicio que proporciona un lugar centralizado para almacenar y gestionar el código fuente de un proyecto de software. Para configurar un servidor de repositorio, se pueden seguir los siguientes pasos:

1.Elegir un sistema de control de versiones: Hay varios sistemas de control de versiones como GitLab, GitHub, Subversion, etc. Elige el que mejor se adapte a tus necesidades.

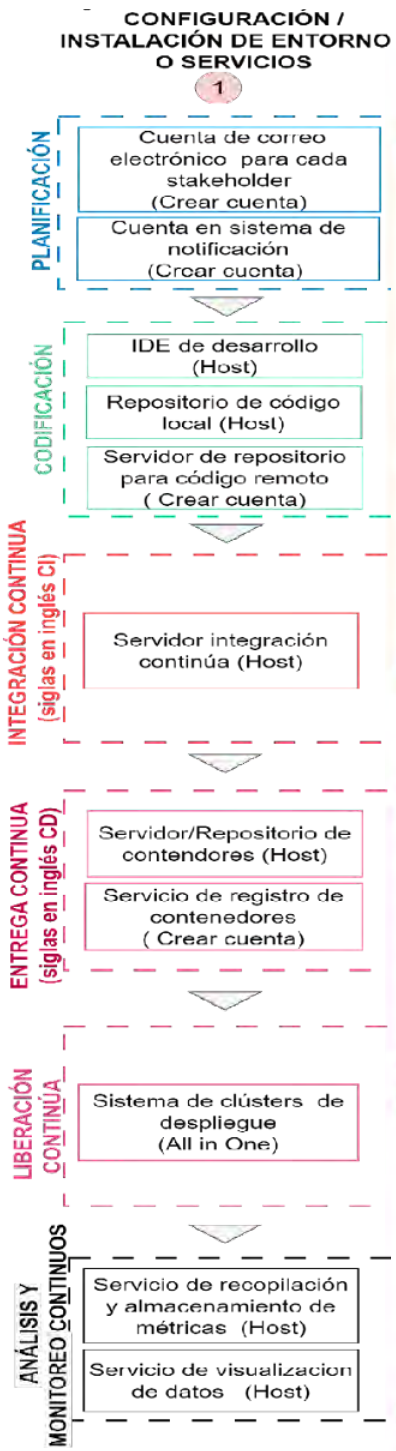
2.Crear una cuenta en el sitio oficial de la aplicación de control de versiones elegido e instalar el software del sistema de control de versiones: Debes instalar el software del sistema de control de versiones en tu Host. Cada sistema tiene su propia documentación de instalación.

3.Configurar el servidor: Después de instalar el software, debes configurar el servidor. Esto incluye configurar los permisos de acceso y la autenticación de usuario.

4.Crear un repositorio: Luego, debes crear un repositorio donde se almacenará el código fuente. Puedes crear múltiples repositorios si lo necesitas.

5.Compartir el repositorio: Una vez que el repositorio está configurado, debes compartir su dirección URL con tu equipo de desarrollo para que puedan acceder a él y colaborar en el código fuente.

6.Mantener el servidor: Finalmente, debes mantener el servidor con las últimas actualizaciones y parches de seguridad para garantizar la seguridad y el rendimiento.



INTEGRACIÓN CONTINUA

Un servicio de integración continua: Es el proceso de integrar, compilar y probar el código fuente de un proyecto de software de manera continua y frecuente. El objetivo principal es detectar y solucionar problemas de manera temprana, reducir los conflictos de integración y garantizar que el software funcione correctamente en todo momento. Para montar un servicio de integración continua se deben seguir los siguientes pasos generales:

1.Seleccionar una herramienta de integración continua, como Jenkins, Travis CI, CircleCI, GitLab CI/CD, entre otras.
2.Configurar el servidor de integración continua, que puede ser local o en la nube, con las Especificaciones necesarias para que pueda ejecutar los scripts y las pruebas correspondientes.
3.Configurar el repositorio de código fuente para que envíe notificaciones al servidor de integración continua cuando se realicen cambios.
4.Crear un archivo de configuración para definir las tareas de integración continua, como compilar el código fuente, ejecutar pruebas unitarias, realizar pruebas de integración, entre otras.
5.Ejecutar las tareas de integración continua automáticamente cada vez que se realicen cambios en el código fuente.
6.Analizar los resultados de las pruebas y notificar a los desarrolladores si se detectan problemas

ENTREGA CONTINUA

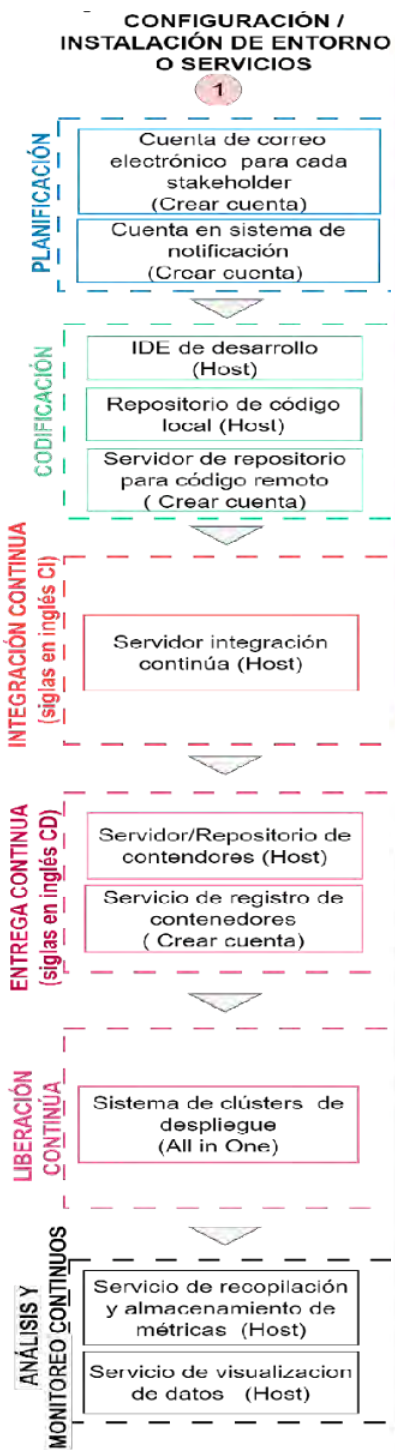
Servidor o repositorio para la entrega continua: La entrega continua es una práctica de desarrollo de software que consiste en automatizar el proceso de entrega del software a producción. Para almacenar y desplegar de forma automatizada las distintas versiones del software, se pueden utilizar varias opciones populares para el servidor o repositorio de entrega continua.

Para establecer un servidor o repositorio para la entrega continua, se debe seleccionar una plataforma de contenedores, como Docker, y proceder con su instalación.

Un servidor de registro de contenedores: Un servidor de registro de contenedores es un componente clave en la gestión y distribución de imágenes de contenedores. Permite almacenar y administrar las imágenes de contenedores utilizadas en entornos de contenerización, como Docker. Para montar un servidor de registro de contenedores para la entrega continua, se pueden seguir los siguientes pasos generales:

1.Crear una cuenta en el proveedor de servicios en la nube elegido y configurar el entorno de contenedores, que puede incluir la creación de una red virtual, la configuración de los nodos, la elección de un motor de orquestación de contenedores, entre otras cosas.

2.Crear una imagen de contenedor personalizada o tomar una imagen existente de un repositorio público.



LIBERACIÓN CONTINUA

Un sistema de clúster: es una infraestructura que agrupa múltiples nodos o servidores para trabajar como una sola entidad coherente. Estos clústeres se utilizan para aumentar la capacidad de procesamiento, mejorar el rendimiento, la disponibilidad y la tolerancia a fallos de una aplicación o sistema. Para montar un sistema de clusters se pueden seguir los siguientes pasos:

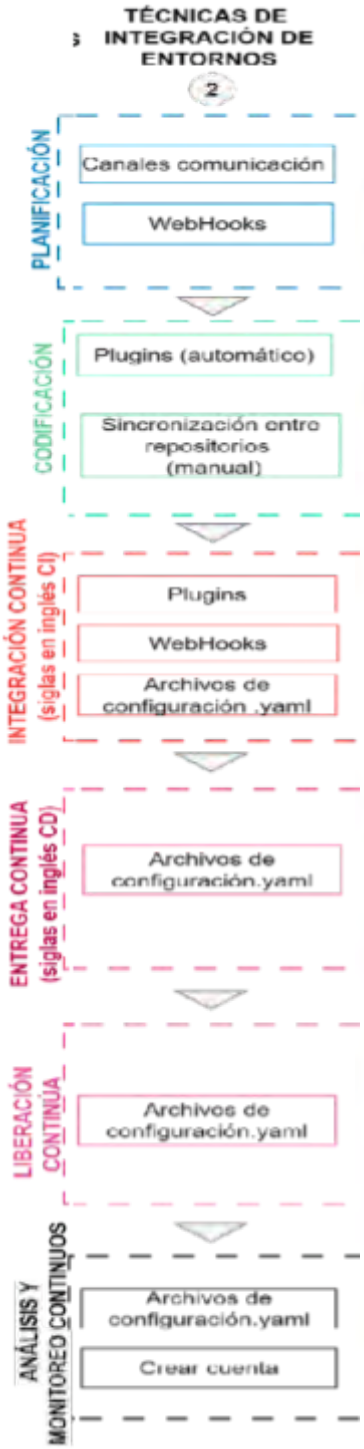
1.Seleccionar una plataforma de orquestación de contenedores, como Kubernetes o Docker Swarm.
2.Crear una cuenta en el proveedor de servicios en la nube y configurar el entorno de contenedores, que puede incluir la creación de una red virtual, la configuración de los nodos, la elección de un motor de orquestacion de contenedores, entre otras cosas.
3.Configurar el archivo de configuración de Kubernetes o Docker Swarm y crear los clústeres.
4.Configurar las políticas de acceso y seguridad para el sistema de clusters, y permitir el acceso a los miembros del equipo que necesiten utilizarlos.
5.Configurar los recursos de cada clúster, como la memoria y el almacenamiento.
6.Configurar los nodos del clúster y asignarles roles específicos, como el de controlador o el de trabajo.
Es aconsejable llevar a cabo una instalación de tipo "All in one" solo si se está en un entorno de prueba o desarrollo, ya que en un entorno de producción se debe utilizar un cluster tradicional donde hay nodos separados que cumplen roles específicos, como nodos de almacenamiento, nodos de procesamiento o nodos de coordinación. En un cluster tipo "All in one" todas esas funciones se concentran en una sola máquina, lo que puede afectar negativamente el rendimiento y la disponibilidad del sistema

ANÁLISIS Y MONITOREO CONTINUO

Un sistema de monitoreo y visualización de clústeres: Un sistema de monitoreo y visualización es una herramienta que permite supervisar y visualizar el estado y el rendimiento de un clúster de manera centralizada. Proporciona información valiosa sobre la salud de los nodos, la utilización de recursos, las métricas de rendimiento y otros aspectos clave del clúster. Esto permite a los administradores y equipos de operaciones monitorear, diagnosticar problemas, tomar decisiones informadas y optimizar el rendimiento.

Para montar un sistema de monitoreo y visualización de clusters, se pueden seguir los siguientes pasos:

1.Seleccionar una plataforma de monitoreo y visualización, como Prometheus o Grafana.
2.Configurar los nodos del clúster para enviar las métricas de la aplicación y del sistema al sistema de monitoreo.
3.Configurar las alertas para detectar y notificar los problemas de la aplicación y del sistema.
4.Configurar la visualización de los datos de monitoreo, como los gráficos y paneles en tiempo real.
5.Ajustar los umbrales de alerta y las políticas de notificación según las necesidades de la aplicación.
6. Configurar las integraciones con otras herramientas, como el sistema de control de versiones o el sistema de tickets.



PLANIFICACIÓN

Desarrollar la lógica de notificación: Se debe desarrollar la lógica de notificación en tu aplicación o sitio web para que envíe notificaciones al sistema elegido. Esto puede incluir la definición de eventos que desencadenan notificaciones y la selección de los usuarios que recibirán las notificaciones.

Integrar los canales de notificación: Los sistemas de notificación pueden enviar notificaciones a través de diferentes canales como notificaciones Push, correos electrónicos, mensajes de texto, entre otros. Debes integrar los canales que desees utilizar y configurarlos adecuadamente.

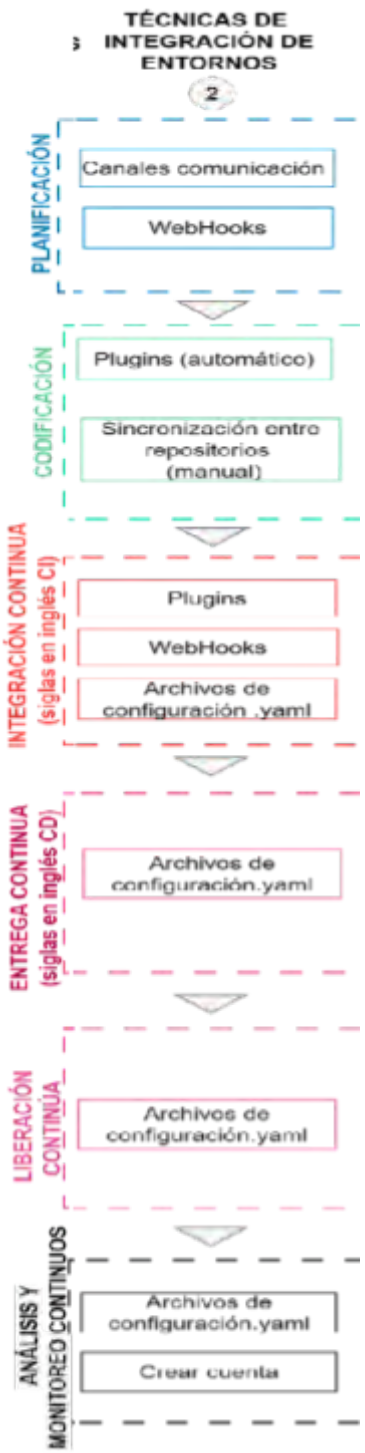
Webhooks (ganchos web): Son una forma de automatizar la integración entre dos aplicaciones web a través de una API (Application Programming Interface).

En términos sencillos, los webhooks son mensajes que se envían de forma automática desde una aplicación a otra cuando se produce un evento o acción determinada en la primera aplicación.

CODIFICACIÓN

Un plugin (complemento): Este término se refiere a un software que se utiliza para añadir características o funcionalidades adicionales a una aplicación existente. En términos generales, un plugin es una pieza de código que se instala e integra en otra aplicación para potenciar su funcionalidad o incorporar nuevas capacidades.

Sincronización a través de Git: Se refiere al proceso de actualizar y mantener alineados los cambios en un repositorio de código compartido por múltiples colaboradores. Git es un sistema de control de versiones distribuido que permite a los desarrolladores trabajar en equipo en proyectos de software de manera eficiente y coordinada.



ENTREGA CONTINUA, LIBERACIÓN CONTINUA Y MONITOREO

Para implementar un sistema de configuración por archivos YAML: Se pueden seguir los siguientes pasos:

1. Crear un archivo YAML que describa la configuración de la aplicación o sistema.

2. Configurar la aplicación o sistema para leer y aplicar la configuración del archivo YAML.

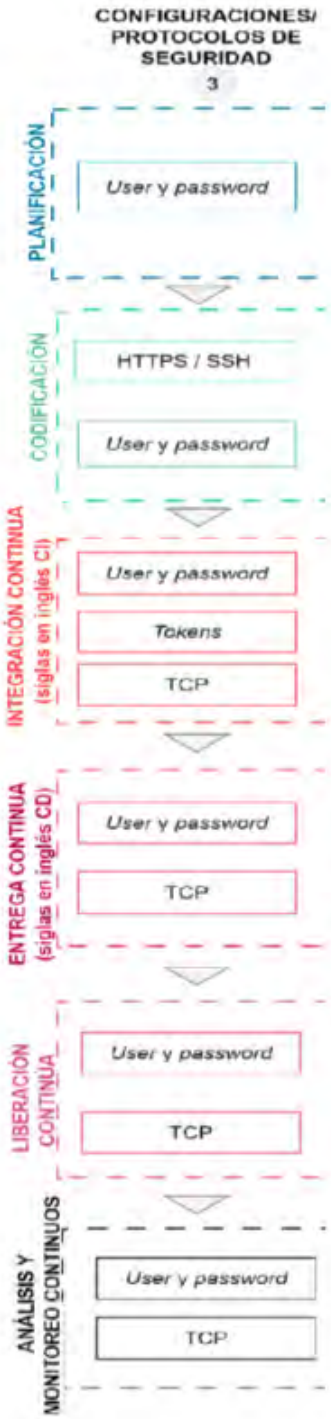
3. Utilizar herramientas de control de versiones, como Git, para gestionar las diferentes versiones de los archivos YAML.

4. Utilizar herramientas de integración y entrega continuas (CI/CD) para automatizar el proceso de implementación de la configuración.

5. Utilizar herramientas de pruebas automáticas para verificar que la configuración se haya aplicado correctamente.

6. Utilizar herramientas de monitoreo para verificar que la aplicación o sistema estén funcionando correctamente después de aplicar la configuración.

En general, un sistema de configuración por archivos YAML es una forma eficiente y escalable de gestionar la configuración de una aplicación o sistema. Además, proporciona una forma consistente y controlada de implementar cambios en la configuración.



PLANIFICACIÓN

Un sistema de seguridad a través de usuario y contraseña: Es un mecanismo común para proteger el acceso a recursos, aplicaciones o sistemas en línea. Este método de autenticación se basa en que cada usuario tenga un nombre de usuario (user) y una contraseña (password) únicos que deben ser proporcionados para acceder a la cuenta o sistema protegido.

CODIFICACIÓN

HTTPS o SSH: Son dos protocolos diferentes que se pueden utilizar para el control de versiones en sistemas de gestión de código fuente, como Git.

Ambos protocolos tienen sus ventajas y desventajas. HTTPS es más fácil de configurar y utilizar en la mayoría de los sistemas, pero puede ser menos seguro que SSH si no se configura correctamente. SSH es más seguro, pero puede ser más difícil de configurar y utilizar para usuarios que no estén familiarizados con claves públicas y privadas. En última instancia, la elección entre HTTPS y SSH depende de las necesidades de seguridad y las preferencias del usuario.

ENTREGA CONTINUA, LIBERACIÓN CONTINUA Y MONITOREO

Un token de seguridad es una forma de autenticación que se utiliza en sistemas de seguridad informática para verificar la identidad de un usuario o de un proceso. Un token de seguridad puede ser un dispositivo físico que se lleva en una llave, un teléfono móvil o una tarjeta inteligente, o bien puede ser un archivo digital que se almacena en un ordenador o dispositivo móvil.

El Protocolo de Control de Transmisión (TCP, por sus siglas en inglés) es uno de los protocolos fundamentales de la capa de transporte en la arquitectura de red TCP/IP.

TCP proporciona una comunicación orientada a conexión y confiable entre aplicaciones que se ejecutan en diferentes sistemas finales (host) de la red. Esto significa que antes de transmitir los datos, se establece una conexión entre el emisor y el receptor, y se realiza un intercambio de paquetes para confirmar que ambos están listos para transmitir y recibir datos.

PLANIFICACIÓN

Sesión de Juego de Rol: Esta es una técnica de compromiso que involucra a las partes interesadas relevantes en una simulación interactiva de las interacciones dinámicas que forman parte del diseño. Se propone la formación de equipos pequeños de 3 a 4 personas como máximo para mejorar la apreciación e involucrar de mejor manera a los alumnos.
Los roles vacantes más importantes en un limitado son: Desarrolladoras, SER (Site Reliability Engineering), Product Manager, DevOps.

Product Manager: Es responsable de definir la visión y estrategia del producto, así como de gestionar su ciclo de vida. Trabaja en estrecha colaboración con el equipo de desarrollo y otras partes interesadas para garantizar que se cumplan los objetivos del producto y se satisfagan las necesidades de los usuarios.

Desarrollador: Es el profesional encargado de escribir y mantener el código de software. Utiliza diferentes lenguajes de programación y herramientas para implementar las funcionalidades y soluciones requeridas. Colabora con otros miembros del equipo para asegurar la calidad del código y el cumplimiento de los requisitos del proyecto.

DevOps: El término "DevOps" se refiere a una cultura y conjunto de prácticas que integran el desarrollo de software (Dev) y las operaciones (Ops). Los profesionales de DevOps son responsables de facilitar la colaboración entre los equipos de desarrollo y operaciones, automatizar y agilizar los procesos de entrega de software, y garantizar una infraestructura confiable y escalable.
SRE (Site Reliability Engineer): Un SRE es un rol enfocado en garantizar la confiabilidad, disponibilidad y rendimiento de los sistemas y servicios de software. Trabajan en estrecha colaboración con los equipos de desarrollo y operaciones para implementar prácticas de ingeniería y automatización que mejoren la fiabilidad y la eficiencia del sistema, y para responder de manera proactiva a problemas e incidentes. Los SRE también se enfocan en la monitorización, el escalado y la planificación de la capacidad de los sistemas.

Selección de metodologías ágiles: Las metodologías ágiles son enfoques iterativos e incrementales para la gestión de proyectos de desarrollo de software. Su objetivo principal es adaptarse a los cambios y responder de manera flexible a los requisitos cambiantes del proyecto, mientras se entregan productos de alta calidad a los clientes. Como SCRUM o Kanban.**Consejo.** Se propone llevar un Sprint 0 donde se proporcionen herramientas para desarrollo.

CODIFICACIÓN

La infraestructura como código: Se alinea con los principios y prácticas de automatización y entrega continua. Permite tratar la infraestructura como cualquier otro componente de software, lo que brinda beneficios en términos de consistencia, reproducibilidad, escalabilidad y agilidad en el despliegue y la gestión de infraestructura. Algunas de las más utilizadas son: AWS CloudFormation, Terraform y Ansible

Decisiones arquitectónicas

1. Uso arquitecturas de software: se refiere a la estructura y organización del software, incluyendo los componentes, las relaciones entre ellos, las decisiones de diseño y las restricciones que guían el desarrollo del sistema. Algunas arquitecturas recomendadas son el uso de microservicios, eventos y contenedores.

Nota. Arquitectura monolítica: Aunque las arquitecturas monolíticas se consideran menos flexibles y escalables en comparación con las arquitecturas basadas en microservicios, aún es posible aplicar los principios y prácticas de DevOps

2. Modelos de diseño de programación: Se refieren a enfoques o paradigmas específicos para desarrollar software, por ejemplo, la programación orientada a objetos (POO), programación estructurada, programación funcional, programación declarativa y programación basada en eventos.

3. Principios y criterios de modularidad: Son pautas que ayudan a lograr un buen modularidad en el diseño y la implementación del software. A continuación, se presentan algunos de los principios y criterios clave de modularidad: principios SOLID, cohesión, acoplamiento, abstracción, encapsulación, reutilización e independencia.

4. Patrón de diseño de software: patrones de diseño proporcionan un enfoque estructurado y probado para abordar desafíos de diseño y mejorar la calidad, el modularidad y la flexibilidad del software. Por ejemplo: Factory, el patrón Adapter y el patrón Observer. Cada uno de estos patrones de diseño tiene una solución específica para un problema común que los desarrolladores pueden aplicar a sus propios proyectos para mejorar la eficiencia y la calidad del código.

La gestión de ramas: La gestión de ramas en el desarrollo de software permite organizar y controlar el flujo de trabajo colaborativo. Dos enfoques comunes son el **Trunk based** development, que se centra en una única rama principal para la entrega rápida de funcionalidades, y el **Git flow**, que utiliza ramas específicas para el desarrollo continuo, versiones y correcciones. La elección depende del contexto y los objetivos del proyecto.

INTEGRACIÓN CONTINUA

Automatización de procesos: Se recomienda seguir los siguientes pasos generales:

1. Identificar los procesos que son susceptibles de automatización: Evalúa qué tareas se realizan de manera repetitiva y manual en tu equipo o empresa y que podrían ser automatizadas.

2. Definir los objetivos de la automatización: Determina qué objetivos se persiguen con la automatización. Por ejemplo, mejorar la eficiencia, reducir los errores humanos, aumentar la productividad, mejorar la calidad, entre otros.

Diseño y configuración de pipelines: crear un plan de implementación: Este diseño puede comenzar en papel y una vez que hayas elegido los procesos, crea un plan de implementación detallado que incluya los pasos a seguir y los plazos para su implementación en el software.

Nota. considera las tareas de las sigúete columna del mismo nivel para la formulación básica de tu pipeline (Build, Test y Merge).

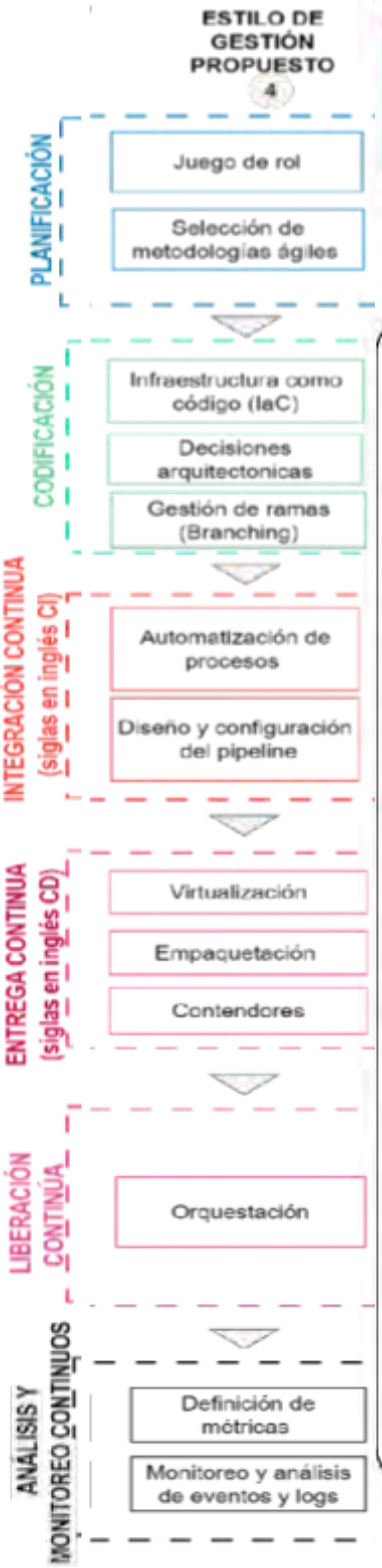
ENTREGA CONTINUA

La **virtualización** es una técnica que crea una capa de abstracción entre el hardware y el sistema operativo, permitiendo la creación de varias máquinas virtuales en un solo servidor físico. Esto es útil para la consolidación de servidores y pruebas aisladas.

La **empaquetación** es el proceso de crear un archivo que contiene todo lo necesario para ejecutar una aplicación. Esto facilita la distribución del software y la creación de imágenes de máquinas virtuales.

Los **contenedores** son una técnica que aisla recursos a nivel del sistema operativo, permitiendo la ejecución de múltiples aplicaciones en un solo sistema operativo. Esta técnica es común en la implementación de aplicaciones en la nube y la orquestación de contenedores.

LIBERACIÓN CONTINUA



La orquestación se utiliza para automatizar y coordinar las diferentes etapas del ciclo de vida de desarrollo de software, desde la compilación y las pruebas hasta el despliegue y el monitoreo.

ANÁLISIS Y MONITOREO CONTINUOS

Definir métricas y monitoreo con análisis de eventos de clusters: puede ser un proceso complejo que requiere una comprensión profunda de la aplicación y la infraestructura subyacente. A continuación, te ofrecemos algunos pasos generales para empezar a definir métricas de monitoreo de clusters:

1. Identifica los componentes críticos del cluster: Determina los componentes del cluster que son más críticos para el correcto funcionamiento de la aplicación, como la capacidad de procesamiento, el uso de memoria, la latencia de red, etc.

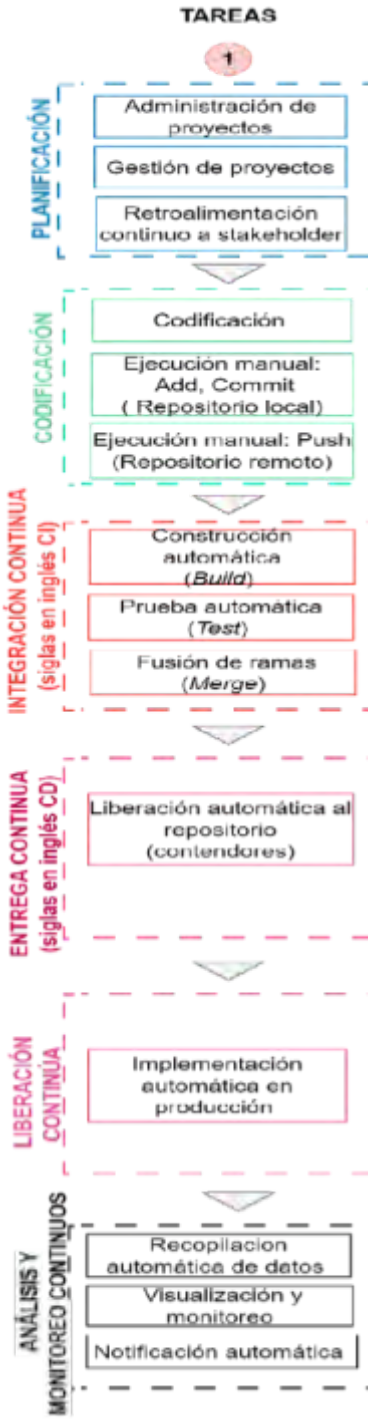
2. Identifica las herramientas de visualización de monitoreo (Dashboard de visualización): Selecciona las herramientas de visualización que utilizarás para capturar las métricas, a partir de que esta nueva herramienta sea compatible con la primera aplicación de captación automática de métricas. Algunas herramientas comunes son Grafana, Zabbix, Nagios, entre otras.

3. Define las métricas: Define las métricas que utilizarás para medir el rendimiento de los componentes críticos del cluster. Por ejemplo, puedes medir la carga de CPU y la memoria utilizada por cada nodo del cluster, la cantidad de solicitudes por segundo, el número de errores, etc.

4. Establece umbrales: Define umbrales para las métricas que te permitan identificar cuando se están acercando a niveles críticos. Por ejemplo, si el uso de memoria de un nodo alcanza el 80%, se puede considerar que está cerca de una situación crítica.

5. Configurar alertas: Configura alertas para que se te notifique cuando se alcance un umbral crítico. Puedes utilizar servicios de notificación como Slack, correo electrónico, SMS, etc.

6. Prueba: Realiza pruebas de carga y otras pruebas para validar que las métricas y umbrales definidos funcionen correctamente. Si es necesario, ajusta las métricas y umbrales para reflejar mejor el rendimiento real del cluster.

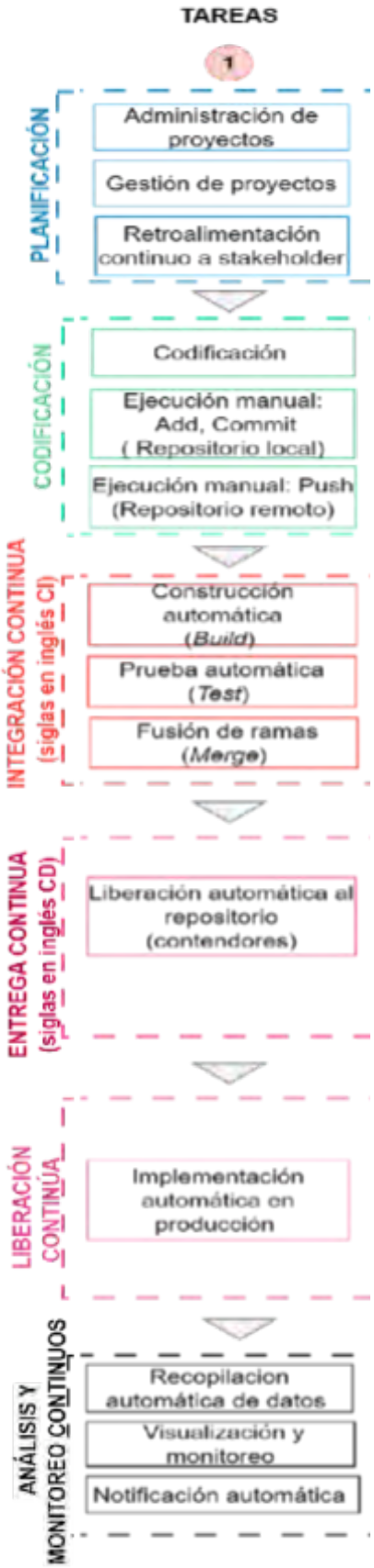


PLANIFICACIÓN

Administración y gestión de proyectos: Para facilitar esta tarea existen varias plantillas para documentar proyectos en DevOps, algunas de las más comunes son:
Plan de proyecto: Este documento incluye información sobre los objetivos del proyecto, los miembros del equipo, los plazos, los recursos y los riesgos asociados con el proyecto.
Documentación de arquitectura: Este documento describe la arquitectura del sistema, incluyendo la infraestructura, los servicios, las aplicaciones y los flujos de datos.
Plan de pruebas: Este documento describe el enfoque de pruebas del proyecto, incluyendo los tipos de pruebas que se realizarán, los criterios de aceptación y las herramientas utilizadas para la automatización de pruebas.
Manual de usuario: Este documento proporciona información sobre cómo utilizar el software o la aplicación desarrollada por el equipo de DevOps.
Documentación de despliegue: Este documento describe los procesos de despliegue del software o la aplicación, incluyendo la configuración del entorno, las herramientas de automatización y las políticas de control de versiones.
Documentación de seguridad: Este documento describe las medidas de seguridad implementadas en el proyecto para proteger los datos y la infraestructura.
Informe de monitoreo y rendimiento: Este documento describe los resultados del monitoreo del sistema y el rendimiento de la aplicación, incluyendo métricas como la disponibilidad, el tiempo de respuesta y el uso de recursos.
Es importante tener en cuenta que cada proyecto es único y puede requerir diferentes tipos de documentación y plantillas. Lo recomendable es adaptar las plantillas existentes a las necesidades específicas del proyecto y del equipo de DevOps.

Gestión de proyectos: Es un proceso continuo, que puede ser automatizado, que identifica, aplica y supervisa los controles para tratar los riesgos de una actividad, proyecto o programa planificado, con el fin de lograr un resultado deseado

Retroalimentación continua (Continuous Feedback): Es un concepto clave en DevOps que se refiere a la práctica de obtener y utilizar comentarios de manera regular durante todo el ciclo de vida del desarrollo y despliegue de software. El objetivo es mejorar continuamente el proceso y el producto final.



CODIFICACIÓN

Codificación: Fin de minimizar la deuda técnica se propone uso de "convenciones de codificación" o "guías de estilo de codificación". Estas reglas son un conjunto de normas y directrices que establecen la forma en que el código debe ser escrito y formateado para garantizar la legibilidad, consistencia y calidad del código. Las convenciones de codificación pueden variar según el lenguaje de programación utilizado y las preferencias de desarrollo de un equipo o comunidad en particular. Algunos ejemplos de guías de estilo de codificación bien conocidas son: PEP 8 (Python Enhancement Proposal 8), Google Java Style Guide y Airbnb JavaScript Style Guide.

Ejecución manual de los comandos add y commit en un repositorio local utilizando Git, sigue estos pasos:

1. Abre una terminal o línea de comandos en tu sistema operativo.

2. Navega hasta el directorio del repositorio local utilizando el comando cd <directorio>, donde <directorio> es la ruta del repositorio en tu sistema.

3. Verifica el estado actual del repositorio ejecutando el comando git status. Esto te mostrará los archivos modificados, agregados o eliminados desde el último commit.

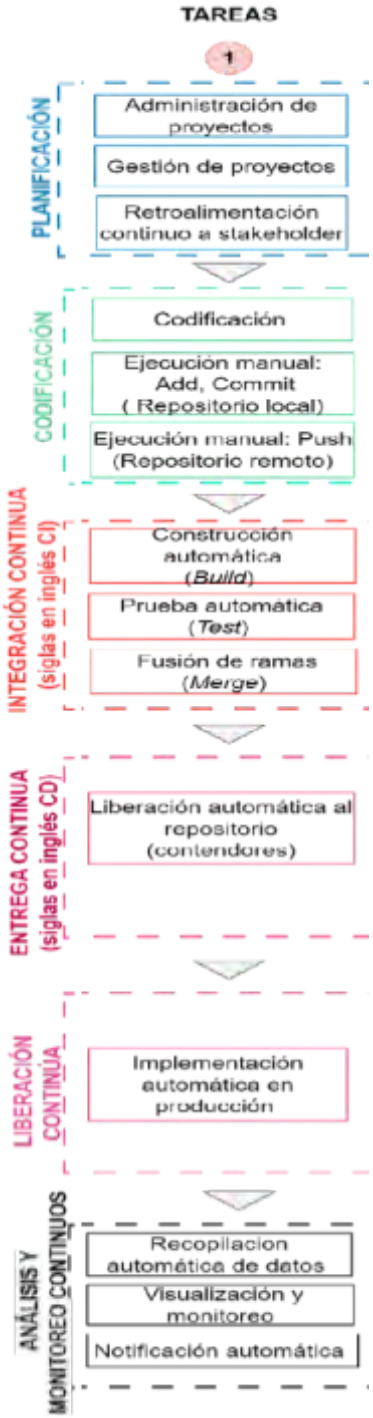
4. Utiliza el comando git add seguido de los nombres de archivo o patrones de archivo para agregar los cambios al área de preparación. Por ejemplo, si deseas agregar todos los archivos modificados, puedes usar git add . o si deseas agregar solo un archivo específico, puedes usar git add <nombre_archivo>.

5. Verifica nuevamente el estado del repositorio con git status para asegurarte de que los cambios se hayan agregado correctamente al área de preparación.

6. Ahora estás listo para realizar el commit. Utiliza el comando git commit -m "mensaje del commit" para crear un nuevo commit con los cambios agregados. Asegúrate de reemplazar "mensaje del commit" con un mensaje descriptivo que explique los cambios realizados.

7. Después de ejecutar el comando git commit, Git creará un nuevo commit con los cambios y les asignará un identificador único. El commit ahora está registrado en el historial del repositorio local.

Recuerda que estos comandos solo afectan al repositorio local. Si deseas sincronizar los cambios con un repositorio remoto, deberás utilizar el comando git push posteriormente.

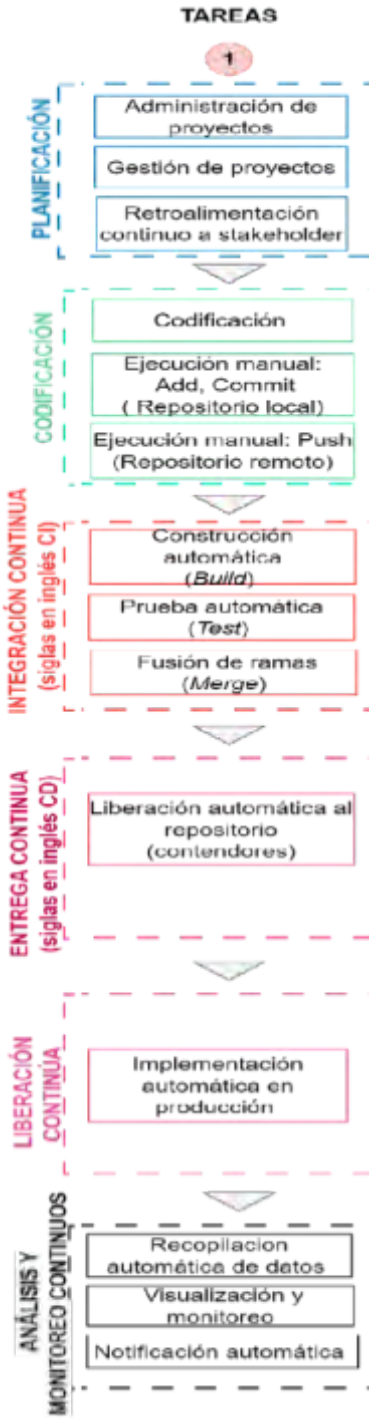


Ejecutar un Push: en un repositorio local utilizando Git, sigue estos pasos, si deseas verificar de manera manual, pero si ya cuentas con la configuración correspondiente del trigger o plugin de Webhook correspondiente, este paso se llevará a cabo de manera automatizada:

Método manual:

1. Abre una terminal o línea de comandos en tu sistema operativo.
2. Navega hasta el directorio del repositorio local utilizando el comando cd <directorio>, donde <directorio> es la ruta del repositorio en tu sistema.
3. Verifica el estado actual del repositorio ejecutando el comando git status. Asegúrate de que tienes los cambios confirmados y estás en la rama correcta para realizar el push.
4. Si no has sincronizado tu repositorio local con el repositorio remoto previamente, puedes agregar el repositorio remoto con el comando git remote add <nombre_remoto> <URL_remoto>. Reemplaza <nombre_remoto> con un nombre descriptivo para el repositorio remoto (por ejemplo, "origin") y <URL_remoto> con la URL del repositorio remoto.
5. Antes de realizar el push, es recomendable ejecutar un pull para obtener las últimas actualizaciones del repositorio remoto. Puedes usar el comando git pull <nombre_remoto> <nombre_rama> para obtener los cambios. Reemplaza <nombre_remoto> con el nombre del repositorio remoto y <nombre_rama> con el nombre de la rama en la que deseas hacer el pull.
6. Una vez que tienes los últimos cambios del repositorio remoto, estás listo para realizar el push. Utiliza el comando git push <nombre_remoto> <nombre_rama> para enviar tus cambios al repositorio remoto. Reemplaza <nombre_remoto> con el nombre del repositorio remoto y <nombre_rama> con el nombre de la rama en la que deseas hacer el push.

Git enviará los cambios al repositorio remoto y mostrará información sobre el progreso del push.
Una vez completado el push, los cambios estarán disponibles en el repositorio remoto.
Recuerda que necesitarás los permisos adecuados para realizar un push en el repositorio remoto.



INTEGRACIÓN CONTINUA

La integración continua (CI, por sus siglas en inglés) es una práctica de desarrollo de software que implica la automatización de las etapas de build (compilación), test (pruebas) y merge (fusionar) en un flujo de trabajo continuo. A continuación, se describen los pasos generales para implementar CI con estas tres etapas:

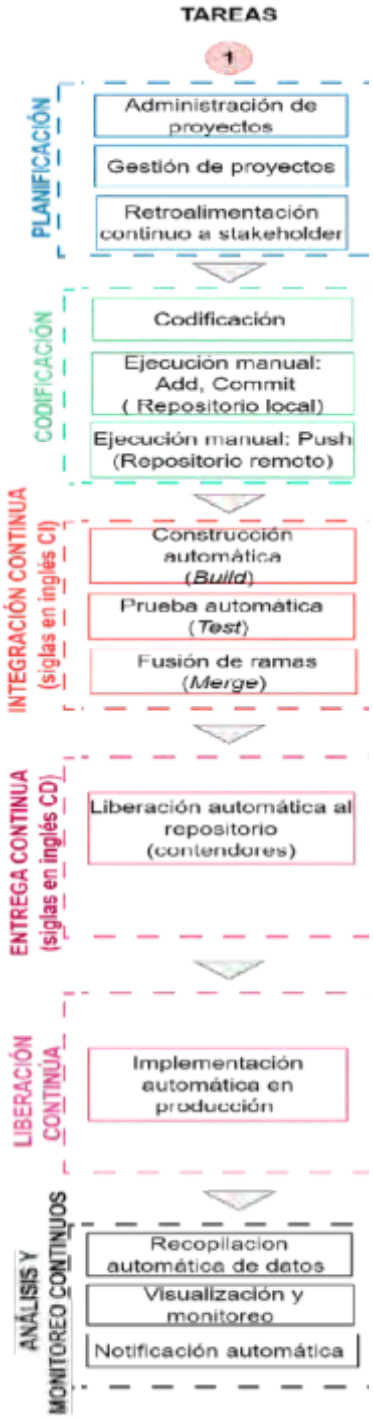
Build: Define un proceso de build automatizado que compile tu código fuente en un artefacto ejecutable. Puedes utilizar herramientas como Maven, Gradle o Make para ejecutar este proceso. Asegúrate de que tu proceso de build se pueda ejecutar automáticamente en el entorno de CI y que produzca el artefacto deseado, como un archivo binario o un paquete de distribución.

Test: Configura una suite de pruebas automatizadas para tu proyecto. Esto puede incluir pruebas unitarias, pruebas de integración, pruebas de rendimiento, etc. Utiliza frameworks y herramientas de pruebas compatibles con tu lenguaje de programación, como JUnit, PyTest o Selenium. Asegúrate de que las pruebas están diseñadas para ejecutarse de forma automatizada y proporcionen resultados claros y significativos, en esta etapa se validan requisitos importantes que darán paso a hacer el Merge o no.

Merge: Establece una política de merge para tu proyecto. Puedes optar por fusionar automáticamente las ramas cuando los pasos de build y test sean exitosos, o puedes requerir **validación** y revisiones manuales antes de la fusión. Configura reglas y flujos de trabajo en tu herramienta de control de versiones para garantizar un proceso de merge coherente y seguro.

Validación: Se sugiere al inicio una validación manual que consiste en comprobar y asegurar que el software cumple con los requisitos y estándares establecidos antes de su implementación. Durante la validación, se realizan pruebas exhaustivas para garantizar que todas las funcionalidades funcionen correctamente y se ajusten a las expectativas del cliente. Además, se verifica la integridad y seguridad del sistema.

Estos son solo los pasos generales para implementar CI con las etapas de build, test y merge. Cada proyecto puede tener sus propias particularidades y requisitos adicionales dependiendo de sus requerimientos.



ENTREGA CONTINUA

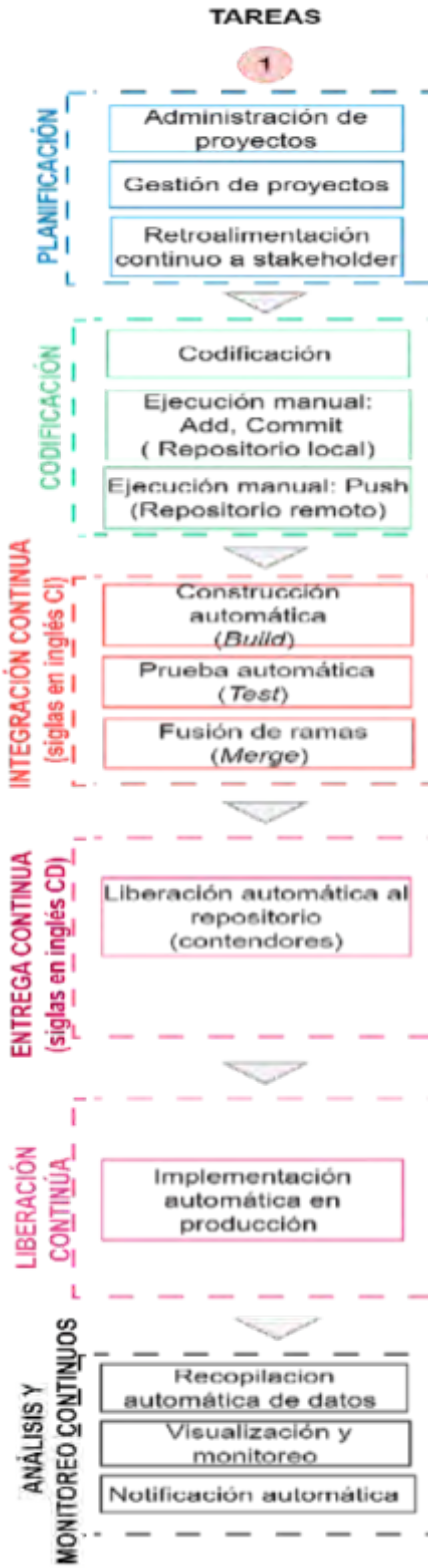
Liberación continua (CD, por sus siglas en inglés): Es una práctica que se relaciona con la integración continua y se refiere a la automatización del proceso de liberación de software a través de un flujo de trabajo continuo. En la liberación continua, los cambios en el código se prueban, empaquetan y despliegan automáticamente en un entorno de producción o de pruebas. A continuación, se describen los pasos generales para implementar la liberación continua: Virtualización, empaquetación y uso de contenedores.

LIBERACIÓN CONTINUA

Implementar automáticamente en producción: Es una práctica avanzada y requiere un enfoque cuidadoso y seguro. Antes de implementar automáticamente en producción, es esencial considerar las implicaciones y riesgos asociados. Aquí hay algunos pasos generales a seguir:

1. Configuración de un entorno de producción: Asegúrate de tener un entorno de producción adecuadamente configurado y listo para recibir la implementación automática. Esto puede incluir la infraestructura necesaria, como servidores, bases de datos y servicios relacionados, así como la configuración de seguridad y las consideraciones de escalabilidad.

2. Pruebas exhaustivas en entornos de pruebas: Antes de implementar en producción, es fundamental realizar pruebas exhaustivas en entornos de pruebas que sean lo más similares posible a la producción. Estas pruebas deben incluir pruebas de funcionalidad, rendimiento, seguridad y cualquier otro aspecto relevante para tu aplicación. Asegúrate de que tus pruebas aborden todos los escenarios posibles y detecten cualquier problema antes de la implementación en producción.



ANÁLISIS Y MONITOREO CONTINUOS

Para recopilar automáticamente datos del clúster, visualizar un panel de control y enviar notificaciones automáticas, puedes seguir los siguientes pasos:

1. Recopilación automática de datos del clúster: Utiliza herramientas de monitorización y recopilación de datos, como Prometheus, Datadog o Elasticsearch, para recopilar métricas y registros del clúster. Configura estas herramientas que recopilan y almacenan automáticamente los datos relevantes del clúster, como el uso de recursos, la salud del sistema y cualquier otro dato que desees monitorizar.

2. Configuración de paneles de control: Utiliza una herramienta de visualización de datos, como Grafana o Kibana, para crear paneles de control personalizados. Estas herramientas te permiten visualizar los datos recopilados de manera intuitiva y comprensible. Configura paneles de control que muestran las métricas y registros clave del clúster de una manera que sea relevante para tu caso de uso.

3. Definición de alertas y notificaciones: Configura reglas de alerta en tus herramientas de monitorización para que detecten situaciones inusuales o problemas en el clúster. Puedes definir umbrales de rendimiento, umbrales de uso de recursos o cualquier otra condición que sea relevante para tus necesidades. Además, configura la entrega de notificaciones automáticas cuando se dispare una alerta, utilizando canales de comunicación como correo electrónico, Slack o servicios de mensajería.

4. Automatización de procesos de notificación: Utiliza scripts o herramientas de automatización, como Prometheus Alertmanager o servicios de notificación en la nube, para gestionar y enviar notificaciones automáticamente. Estas herramientas te permiten definir reglas y acciones específicas para cada tipo de alerta, cómo enviar un correo electrónico a un equipo de operaciones o enviar un mensaje a un canal de Slack.

5. Personalización y ajuste continuo: A medida que obtengas más experiencia y comprendas mejor las necesidades de tu clúster, podrás personalizar y ajustar tus paneles de control, alertas y notificaciones. Realiza un seguimiento constante del rendimiento y la salud del clúster, y realiza cambios en la configuración según sea necesario para optimizar la eficiencia y la detección temprana de problemas.

PLANIFICACIÓN

Pruebas de comunicación: Entre los integrantes de un equipo DevOps son fundamentales para garantizar una colaboración efectiva y un flujo de trabajo fluido. Prácticas para mejorar la comunicación dentro del equipo: Reuniones regulares de equipo, comunicación cara a cara, utilizar herramientas de colaboración en línea, definir canales de comunicación claros, fomentar la transparencia y el intercambio de conocimientos y retroalimentación regular.

CODIFICACIÓN

Pruebas unitarias: Son pruebas que se enfocan en verificar el funcionamiento individual de componentes o unidades de código. Se realizan en un entorno aislado y automatizado. La selección de herramientas de pruebas unitarias debe ser a partir del lenguaje de programación con el que se trabaja. Ejemplo JUnit, NUnit, entre otros.

INTEGRACIÓN CONTINUA

Pruebas de integración: Estas pruebas verifican la interacción y comunicación correcta entre los diferentes componentes y módulos del sistema. Se centran en identificar problemas de integración y asegurar que los componentes trabajen juntos sin problemas.

ENTREGA CONTINUA

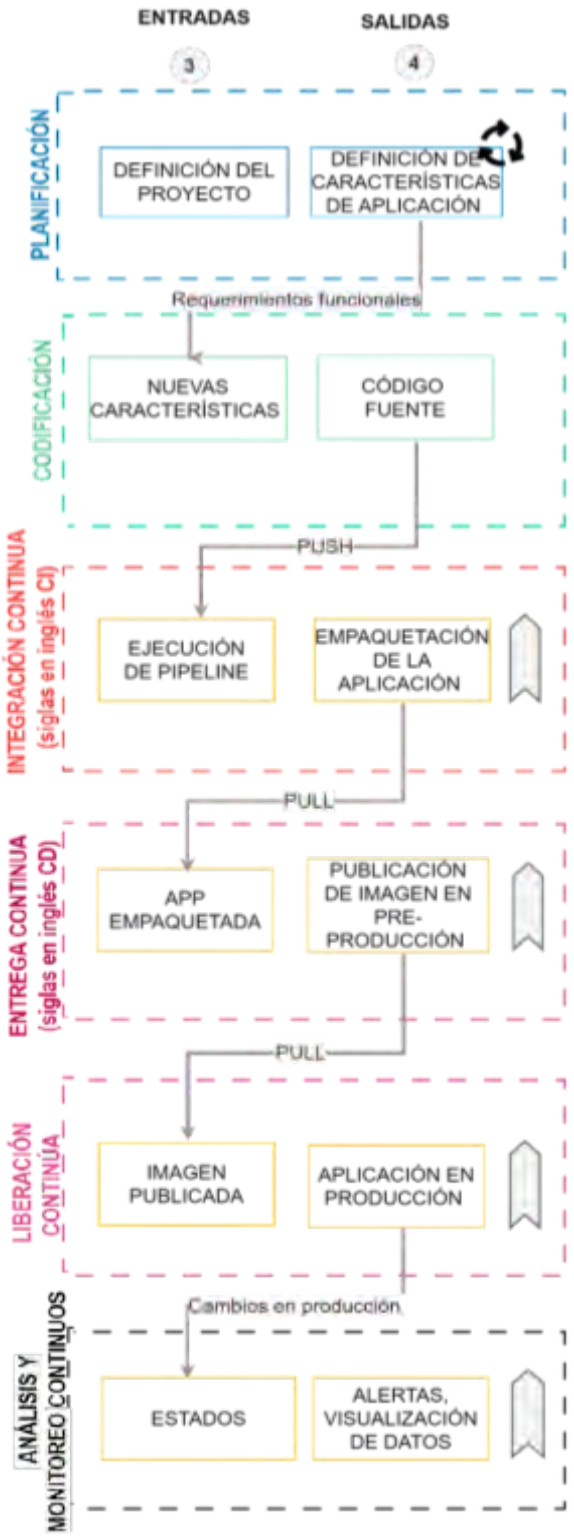
"Contrato de pruebas" o "Contract testing": esta técnica se enfoca en definir y validar los contratos o acuerdos entre las partes involucradas en una comunicación, asegurando que cumplan con las expectativas y requisitos establecidos.

LIBERACIÓN CONTINUA

Pruebas de contrato: Las pruebas de contrato se centran en la interacción entre los componentes de software, por lo general, se realizan a nivel de unidad o componente. El objetivo es validar que las comunicaciones y el flujo de datos entre los componentes se realicen de acuerdo con los contratos definidos.

ANÁLISIS Y MONITOREO CONTINUOS

Pruebas E2E: end-to-end (E2E), también conocida como prueba de extremo a extremo, es un tipo de prueba que se realiza en un sistema o aplicación para validar su funcionamiento completo desde el inicio hasta el final, simulando un escenario de uso real.



PLANIFICACIÓN

A partir de la definición del proyecto, se obtendrá la definición de las características que se formalizarán como un requerimiento funcional. Nota. El recuadro con el símbolo en la esquina superior derecha es el punto de inicio de las interacciones en ciclo de vida DevOps.

CODIFICACIÓN

Una vez definida una nueva característica, se procede a traducirla en código fuente y procedemos a través de un Trigger o un Push da acceso al siguiente nivel.

INTEGRACIÓN CONTINUA

Este nivel ejecuta los pipelines, los cuales generarán feedback al equipo DevOps, la salida de este nivel será la aplicación empaquetada y a través de un Pull automático se accede al siguiente nivel.

ENTREGA CONTINUA

Una vez empaquetada la aplicación, se procede a la publicación de la imagen en los repositorios de imágenes, para tener su disponibilidad en ambientes de preproducción.

LIBERACIÓN CONTINUA

Una vez publicada la imagen y disponible en el ambiente de preproducción en nuestro repositorio, se procede a la publicación de la imagen en un ambiente de producción.

ANÁLISIS Y MONITOREO CONTINUO

A partir de eventos suscitados por los cambios de producción, se deben reflejar con notificaciones en los canales correspondientes para cada uno de los interesados

5.3. Validación de la Arquitectura de Referencia Académica de DevOps (ARA)

La validación de una arquitectura de referencia es un proceso crítico que requiere la opinión y experiencia de profesionales con conocimientos en DevOps y arquitectura de sistemas, como lo describe el artículo [106], trabajo en el que se validó la calidad de su trabajo como herramienta analítica aplicándolo para el análisis de veinticuatro arquitecturas de referencia. Las conclusiones de este trabajo se compararon con las opiniones de expertos/profesionales sobre arquitecturas de referencia documentadas en la literatura. A través de juicios de expertos/profesionales y herramientas de validación automática, se puede garantizar que la arquitectura cumpla con los requisitos necesarios para cumplir su objetivo.

El juicio de expertos a menudo se utiliza durante el proceso de validación de procesos, el juicio y la experiencia se aplican a cualquier detalle técnico y de gestión, según lo describe la Guía del PMBOK.

5.3.1. Diseño de la Validación

Según la Guía del PMBOK, en su cuarta edición: “Una entrevista es una manera formal o informal de obtener información acerca de los interesados, a través de un diálogo directo con ellos. Se realiza habitualmente haciendo preguntas, preparadas o espontáneas, y registrando las respuestas. Las entrevistas se realizan a menudo de manera individualizada, pero también pueden implicar a varios entrevistadores y/o entrevistados. Entrevistar a participantes con experiencia en el proyecto, así como a interesados y expertos en la materia, puede ayudar a identificar y definir las características y funciones de los entregables esperados del proyecto” [101].

Se implementó un proceso de validación para la arquitectura mediante la realización de entrevistas. A través de éstas, se logró capturar y analizar las experiencias, percepciones y comentarios de profesionales en el campo de DevOps. Se diseñó una guía de preguntas estructuradas (Ver **ANEXO E**) que abarcó aspectos fundamentales de la arquitectura ARA, incluyendo sus componentes, prácticas recomendadas y posibles desafíos en su implementación, legibilidad y aplicación potencial.

5.3.2. Selección de Participantes

Se llevó a cabo una cuidadosa selección de participantes, en la que se incluyeron seis personas con amplia experiencia en DevOps, tanto provenientes de la industria como de entornos académicos. La muestra de participantes incluyó profesores, investigadores, profesionales en tecnología de la información y profesionales con experiencia en el ámbito industrial. El objetivo fue obtener una representación diversa que permitiera obtener perspectivas variadas y enriquecedoras en el estudio.

5.4. Metodología de las entrevistas

5.4.1. Desarrollo

En el marco de esta investigación, se llevaron a cabo un total de seis entrevistas, de las cuales cinco se realizaron de manera individual utilizando la plataforma de videoconferencia “Meet” [107]. Se obtuvo el consentimiento de los entrevistados para grabar las sesiones y asegurar la precisión de la información recopilada. Además, se aprovechó la oportunidad de una visita presencial al CENIDET para realizar una entrevista en persona, en la que se estableció un contacto directo y cercano.

Durante cada entrevista, que tuvo una duración aproximada de una hora, se presentó a los participantes la arquitectura de referencia (ARA), así como el enfoque y los objetivos del estudio. Se promovió un ambiente abierto y colaborativo, donde se alentó a los entrevistados a expresarse con sinceridad y a compartir libremente sus opiniones. Este enfoque facilitó la obtención de información valiosa y perspectivas enriquecedoras de los participantes.

5.4.2. Aplicación

Para asegurar la exactitud de las observaciones y simplificar su análisis subsecuente, se documentó cada entrevista realizada. Luego se efectuó un análisis detallado para extraer información pertinente, identificando patrones y tendencias en las respuestas de los entrevistados. Se puso un énfasis particular en las propuestas de mejora y los puntos fuertes de la arquitectura ARA. Para un desglose más completo de este análisis, se puede revisar el **ANEXO F**.

5.5. Temas o áreas exploradas

Durante la entrevista de validación de ARA, se abordaron varios temas relevantes que son fundamentales para comprender y aplicar eficazmente los principios de DevOps en la práctica con esta arquitectura (Ver Fig. 10).

Inicialmente, se abordó la relevancia y creciente adopción de DevOps en la industria del software. Se resaltó su rol en la aceleración de la entrega de software y la mejora de la colaboración. Se citaron casos de organizaciones líderes que han incorporado DevOps exitosamente y los beneficios derivados. Al constatar el auge de DevOps en la industria, el objetivo de este estudio se centró en dotar a los estudiantes de las habilidades y conocimientos necesarios para adaptarse a las dinámicas del mercado laboral. Se subrayó la ventaja que supone para los estudiantes dominar técnicas de automatización y prácticas de DevOps, ya que fomenta el desarrollo de competencias relevantes y podría facilitar su incorporación al mundo laboral.

Se mencionó cómo la adopción de DevOps en el entorno académico puede mejorar la comprensión de los estudiantes sobre los procesos y prácticas actuales de desarrollo de software. Con la familiarización de técnicas de automatización, los estudiantes pueden optimizar procesos de desarrollo, pruebas y despliegue, incrementando su eficiencia y productividad.

En cuanto a DevOps, se profundizó en sus pilares fundamentales, como la integración y entrega continua y la automatización de procesos. Se resaltó cómo DevOps promueve una visión integral del ciclo de vida del desarrollo de software. Asimismo, se exploró la terminología de DevOps, abordando términos como infraestructura como código (IaaC), herramientas de automatización, orquestación de contenedores y gestión del ciclo de vida de las aplicaciones. Se expusieron los desafíos habituales en la implementación de DevOps, como la resistencia al cambio y la necesidad de una cultura y mentalidad adecuadas.

En un segundo enfoque, se abordó la arquitectura de software y sus diferentes tipos, contrastando las arquitecturas de referencia académicas y de la industria. En un tercer enfoque, se exploró la relación entre DevOps y el ámbito académico, discutiendo la manera en que las instituciones educativas están asumiendo el reto de formar a los profesionales del futuro en prácticas y herramientas de DevOps. Se destacaron los esfuerzos por integrar DevOps en la curricula de materias, ofrecer oportunidades de aprendizaje práctico y fomentar la colaboración entre la academia y la industria.

En este marco, se mencionó el papel crucial de ARA al ofrecer una guía y estándares recomendados para implementar DevOps. Se enfatizó cómo ARA puede proporcionar orientaciones sobre la estructura organizativa, las herramientas y las mejores prácticas.

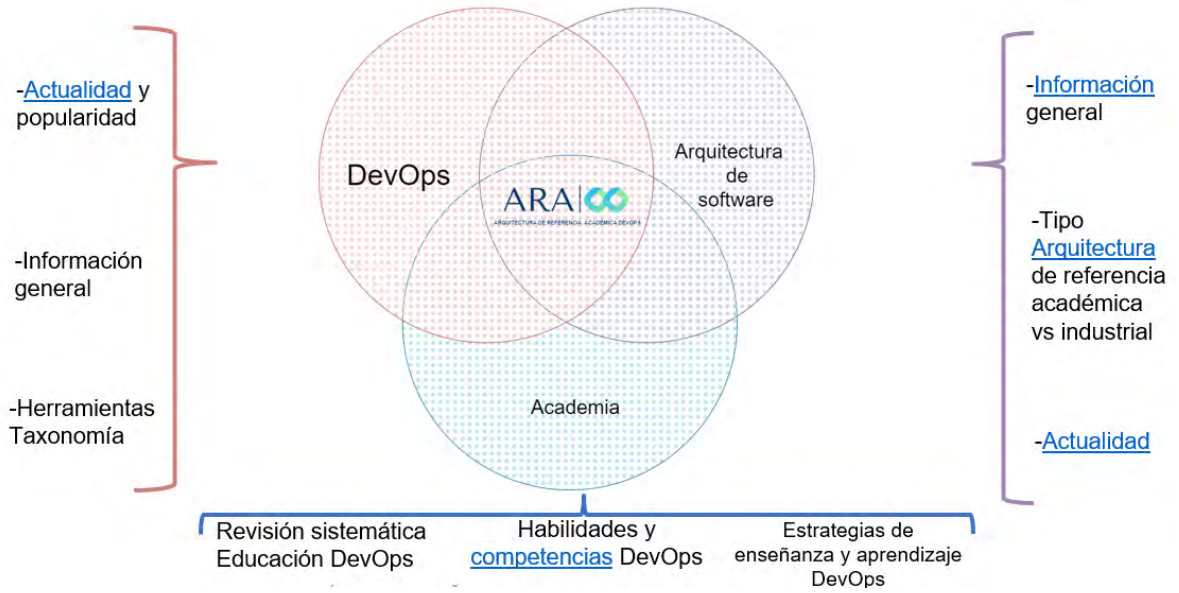


Fig. 10. Temas o áreas exploradas durante las entrevistas.

Nota. La revisión sistemática se puede ver en el anexo G, en el artículo “DevOps desde la academia e industria: una revisión sistemática de la literatura”.

Durante las discusiones, se reconoció la importancia de proteger los derechos de autor de ARA para asegurar su continuidad y promover su difusión en el ámbito académico. Al proteger los derechos de autor, se garantiza la capacidad de continuar publicando investigaciones y trabajos relacionados con ARA, y se brinda la oportunidad de dar a

conocer esta arquitectura a academias y otras instituciones que podrían estar altamente interesadas en su aplicación y estudio.

La protección de los derechos de autor de ARA no solo es fundamental para preservar la integridad de esta arquitectura, sino que también proporciona una base legal sólida para promover su uso responsable y reconocer la labor de aquellos que han contribuido a su desarrollo. Al salvaguardar los derechos de autor, se asegura que ARA se mantenga como una valiosa referencia en el campo de DevOps y que su legado continúe inspirando futuras investigaciones y avances en el área.

Además, la protección de los derechos de autor de ARA ofrece una oportunidad para establecer colaboraciones académicas sólidas. Al dar a conocer la existencia y los beneficios de ARA a academias interesadas, se pueden establecer asociaciones de investigación y programas de estudio que aprovechen la experiencia y el conocimiento acumulados en esta arquitectura de referencia. Esto promoverá la difusión de ARA en entornos académicos y brindará una base sólida para futuras investigaciones y aplicaciones prácticas.

5.6. Análisis de comentarios

A través de esta arquitectura, se han abierto nuevas oportunidades para integrar tecnologías avanzadas, como la inteligencia artificial (IA), en el proceso de toma de decisiones.

ARA podría proporcionar un marco sólido para estandarizar el trabajo de los equipos de desarrollo, operaciones y análisis de manera conjunta y coordinada, a partir de una mejor visualización de su entorno de trabajo, gracias a este esquema.

Hay bastantes campos por cubrir, en temas de seguridad como DevSecOps, la nube, DevOps y la IA, entre otros, que surgieron a partir de estas entrevistas, los cuales pretenden ser considerados más adelante en esta investigación, además de considerar probables colaboraciones con estos expertos.

5.7. Resumen de entrevistas con profesionales/expertos

Se llevaron a cabo entrevistas con diversos expertos en DevOps, quienes desempeñan roles o actividades destacadas en el campo, como se detalla a continuación:

- Dra. Perla Velasco Elizondo: Investigadora candidata a SNI y arquitecta de software, autora del libro "Arquitectura de software: conceptos y ciclo de desarrollo".
- Ing. Enrique Aguilar Martínez: Docente con especialización en arquitectura AWS.
- Dra. Mirna Muñoz: Docente e Investigadora Nivel I del SNI, dirige temas de tesis en DevOps y directora del proyecto de tesis “A guidance to implement or reinforce a DevOps approach in organizations: A case study".
- M. C. Osvaldo Alejandro Hernández León: Desarrollador con especialización en DevOps.
- Ing. Jahaziel Samuel Escobar Glaind: Profesional en SRE (Site Reliability Engineering) DevSecOps.
- Ing. Miguel Ángel Enríquez López: Consultor Senior en DevOps.

La Dra. Velasco Elizondo, gracias a su vasto conocimiento y experiencia en arquitectura de software, realizó una evaluación crítica de la arquitectura propuesta y proporcionó un valioso punto de vista. Su respaldo a la arquitectura fortalece la argumentación sobre su pertinencia y relevancia en el dominio específico.

Los docentes, Ing. Aguilar Martínez y Dra. Muñoz, ofrecieron opiniones sólidas sobre la arquitectura propuesta gracias a su experiencia en investigación y docencia. Su respaldo añade peso a las decisiones de diseño y aporta mayor validez a la propuesta.

El M. C. Hernández León, con su experiencia práctica en DevOps, aportó una visión realista desde el punto de vista de la implementación y operación. Sus comentarios y sugerencias dan respaldo a la viabilidad y aplicabilidad en entornos educativos y de desarrollo basado en el enfoque de DevOps y la ARA.

El Ing. Enríquez, como consultor senior, con su conocimiento actualizado de las últimas tendencias y avances en DevOps, evaluó la arquitectura propuesta desde una perspectiva moderna y relevante. Sus comentarios y observaciones respaldan la elección de enfoques y tecnologías actuales en la arquitectura propuesta.

El Ing. Escobar Glaind, con su experiencia en DevSecOps, evaluó la arquitectura desde la perspectiva de seguridad y desarrollo seguro. Su opinión vanguardista refuerza la inclusión de prácticas de seguridad y protección en el diseño de la arquitectura propuesta.

En resumen, los resultados de las entrevistas con estos expertos y profesionales respaldan la validez preliminar cualitativa de la arquitectura propuesta. Sus comentarios y observaciones indican que la arquitectura es relevante, apropiada y factible en el dominio específico, además de proporcionar sugerencias adicionales para su mejora. Esto fortalece la confianza en la arquitectura propuesta y respalda las decisiones de diseño tomadas.

5.8. Resultados de la validación por juicio de expertos

Obtener una opinión válida de profesionales en DevOps, Desarrollo de Software y Académicos permite conocer la fiabilidad de la arquitectura propuesta. A través de los resultados de entrevistas con dichos profesionales, se podría respaldar la arquitectura en términos de [101], [102], [104]:

Los resultados de las entrevistas son fundamentales para evaluar la idoneidad de la arquitectura propuesta. Los comentarios de los entrevistados brindan una perspectiva externa y realista sobre si la arquitectura es adecuada para abordar los desafíos y requisitos específicos del dominio en cuestión. Los resultados de las entrevistas respaldan la arquitectura propuesta a través del juicio de los profesionales del campo, lo cual es un indicador positivo.

Los comentarios de los entrevistados respaldan la decisión de diseño de la arquitectura al proporcionar evidencia y argumentos sólidos. Los entrevistados expresaron su apoyo a la arquitectura propuesta, por lo tanto, se refuerza la confianza en el diseño de la arquitectura abordada.

Los comentarios de los entrevistados dejan ver que la arquitectura propuesta puede ser beneficiosa para la enseñanza de DevOps. Los comentarios positivos indican que la arquitectura aborda de manera efectiva los problemas y desafíos actuales, lo cual es un factor clave para su éxito. Estas entrevistas dan soporte a la confianza en la arquitectura y su capacidad para cumplir con los objetivos establecidos.

Las entrevistas proporcionaron ideas adicionales para mejorar la arquitectura. Los comentarios y sugerencias de los entrevistados revelaron áreas de mejora y ajustes que no se consideraron inicialmente. Estas ideas adicionales ayudaron a mejorar la arquitectura y abordar aspectos que no se tomaron en cuenta.

Análisis de los resultados de las entrevistas

Resultados de acuerdo a las seis entrevistas y por qué se podría considerar que ARA podría ser idónea y relevante desde el punto de vista de los profesionales que respaldan la toma de decisiones dentro de ARA.

Los comentarios de los entrevistados proporcionaron una perspectiva externa y realista sobre ARA y expresan que es adecuada para abordar algunos de los desafíos y requisitos específicos del dominio DevOps, tales como: evitar que los estudiantes se concentren solo en las herramientas y desvíen su atención del propósito de aprender el paradigma DevOps, identificar información del dominio de manera más clara y precisa, otorgarles un control general del proyecto al funcionar como una guía y patrón genérico académico, involucrar estándares y normas internacionales que otorguen calidad a sus investigaciones o proyectos, y prepararlos para el mundo laboral. Estos son algunos de los retos que, según [8], [13], [14], enfrentan las instituciones y alumnos al emprender en DevOps. Los comentarios indican que la arquitectura ARA podría abordar los problemas y desafíos actuales, lo cual es un factor clave para su posible éxito. Esta validación respalda la confianza en la arquitectura y su capacidad para cumplir con los objetivos establecidos de esta investigación. (ver comentarios, resultados, cambios y propuestas, partir de las entrevistas en **Anexo F**)

Respaldar la toma de decisiones: Los entrevistados expresaron su acuerdo y apoyo a la arquitectura propuesta, lo que refuerza la confianza en las decisiones tomadas.

Identificación de mejoras: Los comentarios y sugerencias de los entrevistados se tomaron en cuenta. Estas ideas adicionales ayudaron a perfeccionar la arquitectura y abordar aspectos que podrían haberse pasado por alto en la etapa de diseño inicial.

En resumen, la validación de una arquitectura de referencia a través de los resultados de las entrevistas podrían respaldar la decisión de diseño. Los comentarios de los entrevistados respaldaron el diseño de ARA y proporcionaron ideas valiosas para mejorarla, lo que en última instancia contribuye al éxito de la implementación y uso de la arquitectura en el dominio DevOps.

5.8.1 Resultados de las entrevistas

Después de analizar las entrevistas y comentarios de expertos/profesionales sobre la arquitectura de referencia ARA en entornos académicos, se identificaron

recomendaciones clave para mejorar y ajustar ciertos aspectos de la arquitectura. Estas recomendaciones incluyen consejos sobre la claridad de las instrucciones, la incorporación de ejemplos prácticos y la adaptación a diversos contextos institucionales. Al seguir estas sugerencias, se podría optimizar aún más la arquitectura y asegurar una mayor efectividad en diferentes escenarios académicos. (Ver comentarios de profesionales en **Anexo F**).

5.8.2. Discusión

Los hallazgos de esta investigación han proporcionado una comprensión de las diferencias entre las arquitecturas de software en el ámbito industrial y académico, así como pautas que ayudan en la definición de estas arquitecturas. Estos hallazgos están directamente relacionados con los objetivos principales de la tesis, que buscaban analizar y comparar las características y los desafíos asociados con las arquitecturas de software en diferentes contextos.

En primer lugar, se identificaron diferencias significativas entre las arquitecturas de software en el entorno industrial y académico. En el ámbito industrial, se observó una mayor orientación hacia la eficiencia, la escalabilidad y la seguridad, mientras que en el ámbito académico prevalecían la experimentación, la flexibilidad y la innovación. Estas diferencias reflejan las necesidades y los objetivos distintos de cada contexto, y resaltan la importancia de adaptar las arquitecturas de software a sus respectivas áreas de aplicación.

Durante esta investigación surgieron desafíos y limitaciones, especialmente en lo que respecta a encontrar y validar la arquitectura con expertos/profesionales no solo de la industria, sino también de la academia y la arquitectura de software. La colaboración y la comunicación entre estos grupos presentaron dificultades debido a sus diferentes enfoques y prioridades. Esto destaca la importancia de establecer un diálogo constructivo y fomentar la colaboración entre los profesionales de diversos campos para lograr una arquitectura de software robusta.

A partir de los hallazgos y las limitaciones identificadas, se proponen estudios y mejoras adicionales en la Arquitectura de Referencia de DevOps. Sería beneficioso realizar investigaciones adicionales que analicen las mejores prácticas de arquitectura de software en diferentes industrias y disciplinas académicas específicas. Además, se podría desarrollar una metodología más precisa y detallada para la validación de arquitecturas, que integre la perspectiva tanto de la industria como de la academia.

Esta investigación ha sido exitosa en el cumplimiento de los objetivos establecidos para la tesis. El objetivo principal fue identificar los principios y procesos clave del paradigma DevOps y luego integrarlos en una estructura básica conceptual con orientación académica. Además, se propuso una Arquitectura de Referencia (ARA) como guía para el desarrollo e implementación de proyectos de software, basada en los principios y procesos identificados.

Durante el proceso de investigación, se llevaron a cabo análisis para identificar los principios y procesos fundamentales de DevOps. Se examinaron diversas fuentes académicas y profesionales, y se realizó una revisión exhaustiva de la literatura existente en el campo de DevOps. Esta investigación rigurosa permitió obtener una comprensión clara de los elementos clave que conforman el paradigma DevOps.

A partir de esta comprensión, se procedió a integrar estos principios y procesos en una estructura básica conceptual orientada a la academia. Se diseñó cuidadosamente la Arquitectura de Referencia (ARA), que sirve como un marco sólido para el desarrollo e implementación de proyectos de software. ARA proporciona una guía práctica y detallada que ayuda a los estudiantes y profesionales a aplicar los principios y procesos de DevOps de manera efectiva en sus proyectos.

Además, para validar los resultados obtenidos, se realizaron entrevistas con expertos y se recopilaron datos cualitativos y se evaluaron para hacer modificaciones para mejorar la efectividad y aplicabilidad de la ARA. Los resultados de la validación, a partir de las entrevistas, confirmaron la validez de la ARA y respaldaron su utilidad como guía en el desarrollo y la implementación de proyectos de software.

En conclusión, esta investigación ha logrado cumplir exitosamente con el objetivo de identificar los principios y procesos clave de DevOps y su integración en una estructura básica conceptual de orientación académica. La propuesta de la Arquitectura de Referencia (ARA) como guía para el desarrollo y la implementación de proyectos de software ha sido validada mediante el juicio de expertos. Estos resultados respaldan, en primera instancia, la utilidad y la aplicabilidad de la ARA en entornos académicos, y establecen una base sólida para futuros avances en el aprendizaje de DevOps.

Capítulo VI

6. Conclusiones y trabajos futuros

6.1. Conclusiones

Se seleccionó un modelo base llamado RAModel y se utilizó la Norma IEEE 42010, una norma internacional que aborda la creación, el análisis y el mantenimiento de arquitecturas, como herramienta de desarrollo adecuada para la Arquitectura de Referencia Académica. Se generó un esquema gráfico conceptual que permitió visualizar los procesos clave de DevOps en el desarrollo y entrega de software, basándose en el Estándar IEEE 2675:2021. Además, se creó una guía de la Arquitectura de Referencia Académica que puede ser utilizada como una guía en el desarrollo e implementación en proyectos de software. En resumen, se logró identificar los principios y procesos clave del paradigma DevOps para integrarlos en una estructura básica conceptual de orientación académica.

Después de validar la Arquitectura de Referencia Académica (ARA) a través del análisis de los comentarios de los entrevistados, se obtuvo una perspectiva externa y realista sobre su capacidad para abordar algunos de los desafíos y requisitos específicos del dominio DevOps. Los comentarios destacan que la ARA puede ayudar a evitar que los estudiantes se concentren solo en las herramientas y desvíen su atención del propósito de aprender el paradigma DevOps, identificar información del dominio de manera más clara y precisa, otorgarles un control general del proyecto al funcionar como una guía y patrón genérico académico, involucrar estándares y normas internacionales que otorguen calidad a sus investigaciones o proyectos, y prepararlos para el mundo laboral. Estos son algunos de los retos que, según [8], [13], [14], enfrentan las instituciones y alumnos al emprender DevOps. Los comentarios indican que la arquitectura ARA podría abordar los problemas y desafíos actuales, lo cual es un factor clave para su posible éxito. En resumen, la validación respalda la confianza en la arquitectura y su capacidad para cumplir con los objetivos establecidos en esta investigación.

Asimismo, y en paralelo al desarrollo de Arquitectura de Referencia Académica (ARA) se lograron importantes hallazgos y se han abordado diversos problemas identificados en el ámbito académico. A continuación, se destacan cómo ARA podría abordar estos problemas como:

La estandarización: ARA podría proporcionar un enfoque estructurado y estandarizado para la enseñanza DevOps en entornos académicos.

Mejora de la reproducibilidad: ARA podría proporcionar una estructura clara y detallada para el diseño y la implementación de DevOps en sistemas académicos.

La Arquitectura de Referencia Académica (ARA) podría usarse como una herramienta que facilite la transferencia de conocimiento para la enseñanza de DevOps. Esto

aceleraría la transferencia de nuevas ideas y técnicas, lo que a su vez promueve el avance y la innovación en el ámbito académico.

Otro hallazgo importante en este trabajo es acerca de la investigación para definir las diferencias entre las arquitecturas de referencia académicas e industriales, lo que resultó en un importante hallazgo. Se identificaron diferencias significativas entre las arquitecturas de referencia académica e industrial, a partir de la recopilación de información general, como se muestra en la Tabla 1. Arquitecturas de referencias académicas vs. industriales del capítulo dos de este documento, presentan sus diferencias principalmente en el enfoque, alcance, fiabilidad y aplicabilidad.

En resumen, se lograron cumplir satisfactoriamente los objetivos planteados en este proyecto. Se seleccionó un modelo adecuado de desarrollo para la arquitectura de referencia académica y se generó un esquema gráfico conceptual que visualiza los procesos clave de DevOps en el desarrollo y entrega de software. Además, se obtuvieron importantes hallazgos sobre las arquitecturas de referencia académicas que puede ser utilizada como referencia en el desarrollo de otras arquitecturas de este tipo. En general, se logró identificar los principios y procesos clave del paradigma DevOps y se integraron en una estructura básica conceptual de orientación académica.

6.2. Trabajos futuros

A continuación, se presentan algunas posibles investigaciones y trabajos futuros relacionados con la arquitectura ARA.

Realizar casos de estudio utilizando ARA en instituciones que ofrecen programas de enseñanza en ingeniería de software para continuar con su validación. Esto permitiría evaluar la efectividad y aplicabilidad de la arquitectura de referencia académica desarrollada en el desarrollo e implementación de proyectos de software en un contexto educativo.

Investigar áreas de mejora para continuar con la madurez de esta arquitectura y buscar extensiones: Identificar áreas de mejora dentro de la arquitectura ARA y buscar posibles extensiones o adaptaciones para abordar problemas específicos puede contribuir a su madurez y evolución continua de ARA.

Comparativos: Realizar estudios comparativos que comparen la ARA con otras arquitecturas y modelos existentes.

Después de la validación de la arquitectura de referencia (AR) mediante la técnica de juicio de expertos/profesionales, se podrían realizar trabajos futuros para medir las características de calidad de la AR. Características de calidad estándar y no estándar, como la flexibilidad, escalabilidad, modularidad, rendimiento y disponibilidad, para ser evaluadas en futuros trabajos.

Referencias

[1] J. Humble y D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, 1st ed. Addison-Wesley Professional, 2010.

[2] J. Davis y R. Daniels, «Effective DevOps».

[3] S. Carey, «What is devops? Transforming software development, explained», InfoWorld. Accedido: 25 de julio de 2023. [En línea]. Disponible en: https://www.infoworld.com/article/3215275/what-is-devops-transforming-software-development.html

[4] «¿Qué es DevOps? Investigación y soluciones», Google Cloud. Accedido: 25 de julio de 2023. [En línea]. Disponible en: https://cloud.google.com/devops?hl=es

[5] I. M. Pereira, T. G. de S. Carneiro, y E. Figueiredo, «Understanding the Context of IoT Software Systems in DevOps». arXiv, 11 de mayo de 2021. Accedido: 27 de julio de 2023. [En línea]. Disponible en: http://arxiv.org/abs/2104.10147

[6] J. Díaz, R. Almaraz, J. Pérez, y J. Garbajosa, «DevOps in practice: an exploratory case study», en *Proceedings of the 19th International Conference on Agile Software Development: Companion*, en XP '18. New York, NY, USA: Association for Computing Machinery, may 2018, pp. 1-3. doi: 10.1145/3234152.3234199.

[7] G. B. Ghantous y A. Gill, «The DevOps Reference Architecture Evaluation : A Design Science Research Case Study», en *2020 IEEE International Conference on Smart Internet of Things (SmartIoT)*, Beijing, China: IEEE, ago. 2020, pp. 295-299. doi: 10.1109/SmartIoT49966.2020.00052.

[8] K. Kuusinen y S. Albertsen, «Industry-Academy Collaboration in Teaching DevOps and Continuous Delivery to Software Engineering Students: Towards Improved Industrial Relevance in Higher Education», en *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, Montreal, QC, Canada: IEEE, may 2019, pp. 23-27. doi: 10.1109/ICSE-SEET.2019.00011.

[9] B. Benni, P. Collet, G. Molines, S. Mosser, y A.-M. Pinna-Déry, «Teaching DevOps at the Graduate Level: A Report from Polytech Nice Sophia», en *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, vol. 11350, J.-M. Bruel, M. Mazzara, y B. Meyer, Eds., en Lecture Notes in Computer Science, vol. 11350. , Cham: Springer International Publishing, 2019, pp. 60-72. doi: 10.1007/978-3-030-06019-0_5.

[10] J.-M. Bruel y M. Jiménez, «DevOps'18 Education Panel: Teaching Feedback and Challenges», en *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, vol. 11350, J.-M. Bruel, M. Mazzara, y B. Meyer, Eds., en Lecture Notes in Computer Science, vol. 11350. , Cham: Springer International Publishing, 2019, pp. 221-226. doi: 10.1007/978-3-030-06019-0_17.

[11] E. Bobrov, A. Bucchiarone, A. Capozucca, N. Guelfi, M. Mazzara, y S. Masyagin, «Teaching DevOps in Academia and Industry: Reflections and Vision», en *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, vol. 12055, J.-M. Bruel, M. Mazzara, y B. Meyer, Eds., en Lecture Notes in Computer Science, vol. 12055. , Cham: Springer International Publishing, 2020, pp. 1-14. doi: 10.1007/978-3-030-39306-9_1.

[12] G. Rong, S. Gu, H. Zhang, y D. Shao, «DevOpsEnvy: An Education Support System for DevOps», en *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE&T)*, Savannah, GA: IEEE, nov. 2017, pp. 37-46. doi: 10.1109/CSEET.2017.17.

[13] M. Fernandes, S. Ferino, A. Fernandes, U. Kulesza, E. Aranha, y C. Treude, «DevOps Education: An Interview Study of Challenges and Recommendations», en *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Software Engineering Education and Training*, may 2022, pp. 90-101. doi: 10.1145/3510456.3514152.

[14] C. Pang, A. Hindle, y D. Barbosa, «Understanding DevOps education with Grounded theory», en *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*, Seoul South Korea: ACM, jun. 2020, pp. 260-261. doi: 10.1145/3377812.3390808.

[15] «IEEE Standard 2675», IEEE Standards Association. Accedido: 29 de mayo de 2023. [En línea]. Disponible en: https://standards.ieee.org/ieee/2675/6830/

[16] R. Cloutier, G. Muller, D. Verma, R. Nilchiani, E. Hole, y M. Bone, «The Concept of Reference Architectures», *Syst. Eng.*, p. n/a-n/a, 2009, doi: 10.1002/sys.20129.

[17] L. Bass, P. Clements, y R. Kazman, *Software architecture in practice*, 3rd ed. en SEI series in software engineering. Upper Saddle River, NJ: Addison-Wesley, 2013.

[18] M. Guessi, F. Oquendo, y E. Y. Nakagawa, «Variability viewpoint to describe reference architectures», en *Proceedings of the WICSA 2014 Companion Volume*, Sydney Australia: ACM, abr. 2014, pp. 1-6. doi: 10.1145/2578128.2578238.

[19] M. Galster, S. Angelov, S. Martínez-Fernández, y D. Tofan, «Reference architectures and Scrum: friends or foes?», en *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, en ESEC/FSE 2017. New York, NY, USA: Association for Computing Machinery, ago. 2017, pp. 896-901. doi: 10.1145/3106237.3117773.

[20] S. Martínez-Fernández, C. P. Ayala, X. Franch, H. M. Marques, y D. Ameller, «Towards guidelines for building a business case and gathering evidence of software reference architectures in industry», *J. Softw. Eng. Res. Dev.*, vol. 2, n.º 1, p. 7, ago. 2014, doi: 10.1186/s40411-014-0007-5.

[21] S. Angelov, P. Grefen, y D. Greefhorst, «A classification of software reference architectures: Analyzing their success and effectiveness», en *2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture*, Cambridge, United Kingdom: IEEE, sep. 2009, pp. 141-150. doi: 10.1109/WICSA.2009.5290800.

[22] P. H. Valle, L. Garcés, M. Guessi, S. Martínez-Fernández, y E. Nakagawa, «Approaches for Describing Reference Architectures: A Systematic Mapping Study», nov. 2020.

[23] S. Martínez-Fernández, C. P. Ayala, X. Franch, y H. M. Marques, «Benefits and drawbacks of software reference architectures: A case study», *Inf. Softw. Technol.*, vol. 88, pp. 37-52, ago. 2017, doi: 10.1016/j.infsof.2017.03.011.

[24] E. Y. Nakagawa, F. Oquendo, y M. Becker, «RAModel: A Reference Model for Reference Architectures», en *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, Helsinki, Finland: IEEE, ago. 2012, pp. 297-301. doi: 10.1109/WICSA-ECSA.212.49.

[25] 14:00-17:00, «ISO/IEC/IEEE 42010:2011», ISO. Accedido: 29 de agosto de 2023.

[En línea]. Disponible en: https://www.iso.org/standard/50508.html
[26] L. E. Lwakatare, P. Kuvaja, y M. Oivo, «Relationship of DevOps to Agile, Lean and Continuous Deployment», en *Product-Focused Software Process Improvement*, vol. 10027, P. Abrahamsson, A. Jedlitschka, A. Nguyen Duc, M. Felderer, S. Amasaki, y T. Mikkonen, Eds., en Lecture Notes in Computer Science, vol. 10027. , Cham: Springer International Publishing, 2016, pp. 399-415. doi: 10.1007/978-3-319-49094-6_27.
[27] M. Hornbeek, «Engineering DevOps».
[28] M. Airaj, «Enable cloud DevOps approach for industry and higher education: Enable cloud DevOps approach for industry and higher education», *Concurr. Comput. Pract. Exp.*, vol. 29, n.º 5, p. e3937, mar. 2017, doi: 10.1002/cpe.3937.
[29] «Blog: Introducción al Ciclo de Vida de DevOps». Accedido: 25 de julio de 2023. [En línea]. Disponible en: https://kranio.io/blog/introduccion-al-ciclo-de-vida-de-devops
[30] Student, Department of CSE, RV College of Engineering, Karnataka, India *et al.*, «Understanding SDLC using CI/CD Pipeline», *Int. J. Soft Comput. Eng.*, vol. 9, n.º 6, pp. 22-25, may 2020, doi: 10.35940/ijsce.F3405.059620.
[31] S. S. Bernie Coyne, *DevOps for Dummies (2nd IBM Limited Edition)*. 2015. Accedido: 29 de mayo de 2023. [En línea]. Disponible en: http://archive.org/details/1460512945Devopsfordummies
[32] J. Blankenship, M. Bussa, y S. Millett, *Pro Agile .NET Development with Scrum*. Berkeley, CA: Apress, 2011. doi: 10.1007/978-1-4302-3534-7.
[33] R. N. Rajapakse, M. Zahedi, M. A. Babar, y H. Shen, «Challenges and solutions when adopting DevSecOps: A systematic review», *Inf. Softw. Technol.*, vol. 141, p. 106700, ene. 2022, doi: 10.1016/j.infsof.2021.106700.
[34] R. Chatley y I. Procaccini, «Threading DevOps Practices through a University Software Engineering Programme», en *2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T)*, Germany: IEEE, nov. 2020, pp. 1-5. doi: 10.1109/CSEET49119.2020.9206211.
[35] L. Greising, A. Bartel, y G. Hagel, «Introducing a Deployment Pipeline for Continuous Delivery in a Software Architecture Course», en *Proceedings of the 3rd European Conference of Software Engineering Education*, Seeon/ Bavaria Germany: ACM, jun. 2018, pp. 102-107. doi: 10.1145/3209087.3209091.
[36] B. P. Eddy *et al.*, «CDEP: Continuous Delivery Educational Pipeline», en *Proceedings of the SouthEast Conference*, Kennesaw GA USA: ACM, abr. 2017, pp. 55-62. doi: 10.1145/3077286.3077301.
[37] R. A. K. Jennings y G. Gannod, «DevOps - Preparing Students for Professional Practice», en *2019 IEEE Frontiers in Education Conference (FIE)*, Covington, KY, USA: IEEE, oct. 2019, pp. 1-5. doi: 10.1109/FIE43999.2019.9028598.
[38] M. Ohtsuki y T. Kakeshita, «Utilizing Software Engineering Education Support System ALECSS at an Actual Software Development Experiment: A Case Study»:, en *Proceedings of the 11th International Conference on Computer Supported Education*, Heraklion, Crete, Greece: SCITEPRESS - Science and Technology Publications, 2019, pp. 367-375. doi: 10.5220/0007723203670375.
[39] neilpeterson, «Browse Azure Architectures - Azure Architecture Center». Accedido: 25 de julio de 2023. [En línea]. Disponible en: https://learn.microsoft.com/en-us/azure/architecture/browse/

[40] «Ejemplos y prácticas recomendadas de arquitecturas de referencia», Amazon Web Services, Inc. Accedido: 12 de agosto de 2023. [En línea]. Disponible en: https://aws.amazon.com/es/architecture/

[41] M. H. Syed y E. B. Fernandez, «A reference architecture for the container ecosystem», en *Proceedings of the 13th International Conference on Availability, Reliability and Security*, Hamburg Germany: ACM, ago. 2018, pp. 1-6. doi: 10.1145/3230833.3232854.

[42] G. B. Ghantous y A. Q. Gill, «DevOps Reference Architecture for Multi-cloud IOT Applications», en *2018 IEEE 20th Conference on Business Informatics (CBI)*, Vienna: IEEE, jul. 2018, pp. 158-167. doi: 10.1109/CBI.2018.00026.

[43] N. F. D. Filho y E. F. Barbosa, «A Contribution to the Establishment of Reference Architectures for Mobile Learning Environments», *IEEE Rev. Iberoam. Tecnol. Aprendiz.*, vol. 10, n.º 4, pp. 234-241, nov. 2015, doi: 10.1109/RITA.2015.2486339.

[44] D. A. Muñoz, H. A. Ordóñez, y V. Buchelli, «Lineamientos para la implementación del modelo CALMS de DevOps en mipymes desarrolladoras de software en el contexto surcolombiano», *Rev. Guillermo Ockham*, vol. 18, n.º 1, pp. 81-91, oct. 2021, doi: 10.21500/22563202.4270.

[45] John Willis, Docker, «Docker and the Three Ways of DevOps», *Docker*, julio de 2015.

[46] P. M. Duvall, S. Matyas, y A. Glover, *Continuous integration: improving software quality and reducing risk*. en Addison-Wesley signature series. Upper Saddle River, NJ: Addison-Wesley, 2007.

[47] G. Kim, J. Humble, P. Debois, y J. Willis, *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution, 2016.

[48] M. Krief, *Learning DevOps: The Complete Guide to Accelerate Collaboration with Jenkins, Kubernetes, Terraform and Azure DevOps*. Packt Publishing, 2019.

[49] «Taxonomía Computacional. Identificación de "Moving Groups". Aplicación a Cúmulos Abiertos». Accedido: 29 de agosto de 2023. [En línea]. Disponible en: http://sedici.unlp.edu.ar/bitstream/handle/10915/19535/080.pdf?sequence=1

[50] R. W. Macarthy y J. M. Bass, «An Empirical Taxonomy of DevOps in Practice», en *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Portoroz, Slovenia: IEEE, ago. 2020, pp. 221-228. doi: 10.1109/SEAA51224.2020.00046.

[51] «Sign in · GitLab», GitLab. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://gitlab.com/users/sign_in

[52] «GitHub: Let's build from here», GitHub. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://github.com/

[53] Atlassian, «Jira | Software de seguimiento de proyectos e incidencias», Atlassian. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://www.atlassian.com/es/software/jira

[54] «Gestiona los proyectos de tu equipo desde cualquier lugar | Trello». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://trello.com/es

[55] Asana, «Software para gestión de proyectos • Asana», Asana. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://asana.com/es/uses/project-management

[56] «Overview - Redmine». Accedido: 2 de octubre de 2023. [En línea]. Disponible

en: https://www.redmine.org/
[57] «Taiga: Your opensource agile project management software». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://taiga.io/es
[58] «Kanban Project Management Software - Kanboard». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://kanboard.org/
[59] «Wekan», SourceForge. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://sourceforge.net/projects/wekan.mirror/
[60] «Git». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://git-scm.com/
[61] Atlassian, «Bitbucket | Git solution for teams using Jira», Bitbucket. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://bitbucket.org/product
[62] «Home · TortoiseSVN». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://tortoisesvn.net/
[63] «Compare, Download & Develop Open Source & Business Software - SourceForge». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://sourceforge.net/
[64] «Mercurial SCM». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://www.mercurial-scm.org/
[65] «Jenkins», Jenkins. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://www.jenkins.io/
[66] «Home – Travis-CI». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://www.travis-ci.com/
[67] «Continuous Integration and Delivery», CircleCI. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://circleci.com/
[68] «CloudBees CodeShip | Software as a Service (SaaS) CI/CD Solution», CloudBees. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://www.cloudbees.com/products/codeship
[69] «The best CI/CD tool for high-performance engineering teams», Semaphore. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://semaphoreci.com/
[70] «Ansible is Simple IT Automation». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://www.ansible.com/
[71] «Puppet Infrastructure & IT Automation at Scale | Puppet by Perforce». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://www.puppet.com/
[72] «Chef Software DevOps Automation Solutions | Chef», Chef Software. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://www.chef.io/
[73] «Terraform by HashiCorp», Terraform by HashiCorp. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://www.terraform.io/
[74] «Saltproject.io». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://saltproject.io/index.html
[75] «Selenium», Selenium. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://www.selenium.dev/
[76] «JUnit 5». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://junit.org/junit5/
[77] «TestNG - Welcome». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://testng.org/doc/

[78] «BDD Testing & Collaboration Tools for Teams | Cucumber». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://cucumber.io/
[79] «Robot Framework». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://robotframework.org/
[80] «Appium Documentation - Appium Documentation». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: http://appium.io/docs/en/2.1/
[81] «JavaScript Component Testing and E2E Testing Framework | Cypress». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://www.cypress.io/
[82] «The World's Most Popular API Testing Tool | SoapUI». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://www.soapui.org/
[83] Prometheus, «Prometheus - Monitoring system & time series database». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://prometheus.io/
[84] «Grafana: The open observability platform», Grafana Labs. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://grafana.com/
[85] «Nagios Open Source | Nagios Open Source». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://www.nagios.org/
[86] «Zabbix :: The Enterprise-Class Open Source Network Monitoring Solution». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://www.zabbix.com/index
[87] «Icinga » Monitor your entire Infrastructure with Icinga», Icinga. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://icinga.com/
[88] «Cacti® - The Complete RRDTool-based Graphing Solution». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://www.cacti.net/
[89] «Software del Portal de Clientes | Productos Sensus», Sensus - Latin America and Caribbean. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://sensus.com/lar/es/products/customer-portal-software/
[90] «M/Monit». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://mmonit.com/
[91] Datadog, «Cloud Monitoring as a Service | Datadog», Cloud Monitoring as a Service. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://www.datadoghq.com/
[92] «Splunk | The Key to Enterprise Resilience», Splunk. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://www.splunk.com
[93] «Production-Grade Container Orchestration», Kubernetes. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://kubernetes.io/
[94] «Use Docker Compose», Docker Documentation. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://docs.docker.com/get-started/08_using_compose/
[95] «Vagrant by HashiCorp», Vagrant by HashiCorp. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://www.vagrantup.com/
[96] «Open Source Cloud Computing Infrastructure», OpenStack. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://www.openstack.org/
[97] «TeamCity: la herramienta de CI/CD de JetBrains, sin complicaciones», JetBrains. Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://www.jetbrains.com/es-es/teamcity/
[98] Atlassian, «Bamboo: servidor de integración continua y compilación», Atlassian. Accedido: 2 de octubre de 2023. [En línea]. Disponible en:

https://www.atlassian.com/es/software/bamboo
[99] «Concourse CI». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://concourse-ci.org/
[100] «Open Source Continuous Delivery and Release Automation Server | GoCD». Accedido: 2 de octubre de 2023. [En línea]. Disponible en: https://www.gocd.org/index.html
[101] «Unknown - FUNDAMENTOS PARA LA DIRECCIÓN DE PROYECTOS (GUÍA DEL PMBOK ®) Cuarta edición.pdf».
[102] J. Kontio, J. Bragge, y L. Lehtola, «The Focus Group Method as an Empirical Tool in Software Engineering», en *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, y D. I. K. Sjøberg, Eds., London: Springer London, 2008, pp. 93-116. doi: 10.1007/978-1-84800-044-5_4.
[103] M. N. Rajasekaran y D. S. M. Jagatheesan, «A Delphi based Expert Judgment Techniques applied with Capability Maturity Model Integration to validate the Agile Scrum based MVP Architecture Framework for Android Mobile Application Development», vol. 20, n.º 16.
[104] C. Orozco y C. Pardo, «Modelo de métricas para apoyar la evaluación de DevOps en empresas de desarrollo de software», 2022. doi: 10.13140/RG.2.2.19963.59680.
[105] M. E. García-Ruiz y F. J. Lena-Acebo, «Aplicación del metodo delphi en el diseño de una investigación cuantitativa sobre el fenómeno FABLAB», *Empiria Rev. Metodol. Cienc. Soc.*, n.º 40, p. 129, may 2018, doi: 10.5944/empiria.40.2018.22014.
[106] S. Angelov, P. Grefen, y D. Greefhorst, «A framework for analysis and design of software reference architectures», *Inf. Softw. Technol.*, vol. 54, n.º 4, pp. 417-431, abr. 2012, doi: 10.1016/j.infsof.2011.11.009.
[107] «Google Meet». Accedido: 4 de octubre de 2023. [En línea]. Disponible en: https://meet.google.com/