UNIVERSIDAD MAYOR, REAL Y PONTIFICIA DE
SAN FRANCISCO XAVIER DE CHUQUISACA

VICERRECTORADO

CENTRO DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

FACULTAD DE CIENCIAS Y TECNOLOGÍA

“AUTOMATIZACIÓN DE DESPLIEGUE DE UN SISTEMA DESARROLLADO BAJO LA ARQUITECTURA DE MICROSERVICIOS”

TRABAJO EN OPCIÓN A DIPLOMADO EN DEVOPS

Ginelda Villca Mercado

Sucre, Bolivia

2024

CESIÓN DE DERECHOS

Al presentar este trabajo como requisito previo para la obtención del Diploma en DevOps V1 de la Universidad Mayor, Real y Pontificia de San Francisco Xavier de Chuquisaca, autorizo al Centro de Estudios de Posgrado e Investigación o a la Biblioteca de la Universidad, para que se haga de este trabajo un documento disponible para su lectura según normas de la Universidad.

También cedo a la Universidad Mayor, Real y Pontificia de San Francisco Xavier de Chuquisaca, los derechos de publicación de este trabajo o parte de él, manteniendo mis derechos de autor hasta un periodo de 30 meses posterior a su aprobación.

Ginelda Villca Mercado

Sucre, junio de 2024

DEDICATORIA

Esta monografía se la dedico con profundo agradecimiento a Dios, quien ha sido mi fuente de fortaleza y sabiduría en cada paso de este camino académico.

A mis padres en el cielo, Ricardo y Balbina, quienes han sido mis pilares y guías eternos, ejemplos de perseverancia y dedicación, además de haberme brindado su amor incondicional en cada etapa de mi vida. A mis hermanos: Luis, Henry y Javier, por su apoyo constante han sido fundamentales en este camino.

AGRADECIMIENTOS

Con gratitud infinita, deseo expresar mis más sinceros agradecimientos a la fuente de todo Dios, por ser mi guía en cada paso de este viaje académico. A mis padres, les debo todo lo que soy y puedo llegar a ser; por su apoyo incondicional, sus sacrificios y su constante aliento. A mis hermanos, por su presencia reconfortante y su ánimo en cada paso. A mis amigos, quienes han sido mi sostén en los momentos difíciles y han compartido conmigo cada logro y desafío. A mis estimados docentes, cuya dedicación y orientación han sido fundamentales en mi formación académica y personal. A mi docente Doriz Thenier por toda la paciencia. A mi tutor personal Marco Sánchez. Su sabiduría y apoyo han sido una luz en mi camino.

RESUMEN

El presente trabajo de investigación tiene como objetivo principal determinar un conjunto de prácticas y estrategias para el despliegue efectivo de microservicios, integrando prácticas de DevOps, con el fin de mejorar la eficiencia, la calidad y la consistencia del ciclo de vida del desarrollo de software. La automatización del despliegue de sistemas es un aspecto fundamental en el desarrollo de software, ya que permite ahorrar tiempo y recursos, así como mejorar la eficiencia y la calidad de los procesos. Por lo anteriormente señalado la arquitectura de microservicios ha ganado popularidad en los últimos años debido a su capacidad para facilitar el desarrollo, la implementación y el mantenimiento de sistemas complejos. Sin embargo, el despliegue de sistemas basados en esta arquitectura puede resultar complicado y propenso a errores si no se cuenta con un proceso adecuado de automatización.

En este sentido, el presente trabajo investigativo apoyándose en la teoría de sistemas determina las mejores prácticas y estrategias para el despliegue efectivo de microservicios junto con prácticas DevOps, asimismo se optó por la plataforma de Microsoft Azure para documentar la implementación básica de sus servicios, antes de poder realizar la integración de microservicios con la filosofía DevOps. El objetivo es llenar todos los prerrequisitos de la plataforma antes de realizar el objetivo mismo de la presente investigación.

En conclusión, la automatización del despliegue de sistemas desarrollados bajo la arquitectura de microservicios es un aspecto crucial para garantizar la eficiencia y la calidad de los procesos de desarrollo de software. Las prácticas y estrategias para el despliegue efectivo de microservicios, propuesto en este trabajo ofrece una forma estructurada y eficaz de implementar esta automatización.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE CUADROS

INTRODUCCIÓN

1. Antecedentes

En la era actual de la tecnología de la información, los microservicios son tanto un estilo de arquitectura como un modo de programar software. Con los microservicios, las aplicaciones se dividen en sus elementos más pequeños e independientes entre sí. A diferencia del enfoque tradicional y monolítico de las aplicaciones, en el que todo se compila en una sola pieza, los microservicios son elementos independientes que funcionan en conjunto para llevar a cabo las mismas tareas. Cada uno de esos elementos o procesos es un microservicio. Este enfoque de desarrollo de software valora el nivel de detalle, la sencillez y la capacidad para compartir un proceso similar en varias aplicaciones. Es un elemento fundamental de la optimización del desarrollo de aplicaciones hacia un modelo nativo de la nube.

Sin embargo, junto con estas ventajas, los sistemas basados en microservicios también presentan una serie de desafíos únicos, especialmente en lo que respecta a la entrega de software. La entrega continua de software, que implica la automatización de los procesos de desarrollo, prueba y despliegue para permitir entregas frecuentes y consistentes, ha surgido como un enfoque fundamental para abordar este desafío. En este contexto, la cultura DevOps, que promueve una estrecha colaboración entre los equipos de desarrollo y operaciones, ha demostrado ser un habilitador clave para la mejora de la entrega continua de software.

En el corazón de la cultura DevOps se encuentra el concepto de automatización, donde las tareas manuales repetitivas se reemplazan con herramientas y procesos automatizados. La automatización del despliegue, juega un papel crucial en la entrega continua de software al permitir la implementación rápida y confiable de cambios en el entorno de producción. Al automatizar el proceso de despliegue, las organizaciones pueden reducir el riesgo de errores humanos, acelerar los tiempos de entrega y mejorar la estabilidad y la consistencia del entorno de producción.

Los microservicios tienen el fin de crear un sistema modular esto trae consigo las metodologías ágiles que permiten a los equipos trabajar de manera independiente. Uno de los problemas que pueden generarse es que los equipos estén cada vez más aislados unos de otros y tengan una poca o nula comunicación, lo que generará problemas en el desarrollo del sistema entero, ya que, al igual que los microservicios, los equipos deben trabajar juntos para conseguir un

objetivo (Fowler, 2016). En 1968 Melvin E. Conway realizó un estudio en el cual determinó que “las organizaciones que diseñan sistemas están limitadas a producir diseños que son copias de las estructuras de comunicación de dichas organizaciones” (Conway, 1968). También existe evidencia de distintas organizaciones que migraron de monolito a microservicios, se detectó que uno de los desafíos más importantes eran los organizacionales debido a que los equipos tuvieron que hacer frente a un cambio de mentalidad al tener que establecer procesos ágiles (Fritzsch et al., 2019). De igual forma reportaron que la capacitación de personal es una tarea que consume mucho tiempo y si a eso se le añade la flexibilidad que promueve la arquitectura en cuanto a las tecnologías de desarrollo es posible que la coordinación entre diferentes equipos se vuelva compleja.

2. Justificación

La automatización del despliegue de microservicios es una parte importante de la implementación de DevOps. El despliegue manual de microservicios es un proceso complejo y propenso a errores. La automatización del proceso ayuda a reducir errores, mejorar la eficiencia y acortar el tiempo de implementación. Es por ello que automatizar el despliegue de microservicios se considera crucial por las siguientes razones:

2.1. Justificación Operativa.

Permite realizar despliegues de manera más rápida y eficiente que los métodos manuales. Garantiza que el proceso de despliegue sea consistente en todos los entornos, desde desarrollo hasta producción. Esto reduce la posibilidad de errores humanos y asegura que cada despliegue siga las mismas prácticas y estándares, lo que mejora la calidad del software y facilita la resolución de problemas.

Automatizar el despliegue de microservicios ayuda a reducir los costos operativos al minimizar el tiempo y los recursos necesarios para implementar y mantener el software. Además, al reducir la posibilidad de errores y problemas durante el despliegue, se disminuyen los costos asociados con la resolución de incidentes y la pérdida de productividad.

2.2. Justificación metodológica.

La justificación metodológica de la automatización del despliegue continuo en microservicios se fundamenta en una serie de beneficios y necesidades operativas y estratégicas que son cruciales para el éxito de proyectos de software en entornos complejos y dinámicos. A continuación, se presentan las principales razones para implementar esta metodología:

1. **Mejora de la Calidad del Software.** La automatización del despliegue continuo permite la integración y entrega frecuente de cambios en el código, lo que facilita la detección temprana de errores. Esto se traduce en una mejora significativa en la calidad del software, ya que los problemas se identifican y corrigen rápidamente, antes de que se acumulen y se conviertan en problemas mayores.
2. **Reducción del Tiempo de Entrega**. La automatización de procesos permite realizar despliegues de manera rápida y eficiente, reduciendo el tiempo necesario para llevar una nueva funcionalidad o una corrección de errores desde el desarrollo hasta la producción. Esto es especialmente crítico en un entorno de microservicios, donde los componentes del sistema son independientes y pueden necesitar ser actualizados con frecuencia.
3. **Consistencia y Confiabilidad.** La automatización elimina la variabilidad y los errores humanos en los procesos de despliegue. Esto asegura que cada despliegue se realice de manera consistente, siguiendo los mismos pasos y estándares, lo que aumenta la confiabilidad del sistema y minimiza el riesgo de fallos relacionados con el despliegue.
4. **Escalabilidad.** En un entorno de microservicios, el sistema puede crecer rápidamente en complejidad a medida que se añaden nuevos servicios. La automatización del despliegue continuo facilita la gestión de esta complejidad, permitiendo escalar los procesos de desarrollo y despliegue sin necesidad de incrementar proporcionalmente el esfuerzo manual.
5. **Feedback Rápido.** El despliegue continuo automatizado proporciona un ciclo de feedback rápido, permitiendo a los desarrolladores obtener información casi inmediata sobre el estado de sus cambios en un entorno de producción. Esto permite ajustes y mejoras continuas, lo que es vital para la innovación y la respuesta ágil a los cambios en los requisitos del mercado.

6. **Costos Reducidos.** Aunque la implementación inicial de un sistema automatizado puede ser costosa, a largo plazo, la automatización reduce los costos operativos al disminuir la necesidad de intervenciones manuales, reducir el tiempo de inactividad y mejorar la eficiencia del equipo de desarrollo.
7. **Facilitación de la Colaboración y la Integración.** En un entorno de microservicios, diferentes equipos pueden trabajar en diferentes componentes del sistema. La automatización del despliegue continuo facilita la integración de los diferentes servicios, asegurando que todos los componentes funcionen correctamente juntos y permitiendo una colaboración más efectiva entre los equipos.
8. **Cumplimiento y Auditoría.** Los procesos automatizados son más fáciles de documentar y auditar, lo que facilita el cumplimiento de normativas y estándares de la industria. La capacidad de rastrear y reproducir cada despliegue contribuye a una mayor transparencia y responsabilidad.
9. **Reducción del Riesgo.** La capacidad de desplegar cambios de manera incremental y con pruebas automatizadas reduce el riesgo de introducir fallos en producción. Además, en caso de errores, los procesos automatizados suelen incluir mecanismos de rollback que permiten revertir rápidamente a un estado estable.

3. Situación problémica.-

El despliegue manual de microservicios enfrenta una serie de desafíos y problemas comunes que pueden afectar la eficiencia, la calidad y la estabilidad del proceso. Algunos de estos desafíos son:

1. **Errores humanos.-** Los procesos manuales de despliegue son propensos a errores humanos, como la omisión de pasos críticos, configuraciones incorrectas o inconsistencias entre los entornos de desarrollo y producción. Esto puede llevar a fallos en el sistema, tiempos de inactividad y una mayor carga de trabajo para el equipo de desarrollo y operaciones.
2. **Tiempo de entrega lento.-** El despliegue manual de microservicios es un proceso lento y laborioso, que puede retrasar la entrega de nuevas funcionalidades y correcciones de errores. En un entorno competitivo, esta lentitud puede significar una desventaja

significativa, impidiendo a la organización responder rápidamente a las necesidades del mercado y de los usuarios.

3. **Dificultad para escalar.-** A medida que el número de microservicios y la complejidad del sistema crecen, gestionar los despliegues de forma manual se vuelve impracticable. La falta de escalabilidad en los procesos de despliegue puede llevar a cuellos de botella operativos y a una mayor probabilidad de fallos.
4. **Falta de consistencia.-** Sin automatización, es difícil garantizar que cada despliegue se realice de manera consistente. Las variaciones en los procedimientos pueden causar que el software funcione de manera diferente en diferentes entornos, lo que complica la resolución de problemas y la garantía de calidad.
5. **Feedback lento y poca visibilidad.-** Los procesos manuales dificultan la obtención de feedback rápido sobre el estado del software en producción. Esto puede retrasar la identificación y corrección de errores, y reducir la capacidad del equipo para mejorar el software de manera continua y ágil.
6. **Riesgo alto de fallos en producción.-** La falta de pruebas automatizadas y despliegues incrementales incrementa el riesgo de que los cambios introducidos en producción causen fallos. La ausencia de mecanismos automáticos de rollback dificulta la recuperación rápida ante problemas, prolongando los tiempos de inactividad y afectando negativamente la experiencia del usuario.
7. **Mayor costo operativo.-** La dependencia de procesos manuales incrementa los costos operativos debido a la necesidad de más personal y tiempo para realizar tareas repetitivas y propensas a errores. Además, la resolución de problemas y la corrección de fallos resultan más costosas y laboriosas.
8. **Dificultades en la colaboración y la integración.-** En un entorno de microservicios, diferentes equipos pueden estar trabajando en diferentes componentes simultáneamente. Sin una automatización adecuada, la integración de estos componentes puede ser compleja y propensa a errores, dificultando la colaboración y sincronización efectiva entre equipos.
9. **Cumplimiento y auditoría complicados.-** La falta de procesos automatizados dificulta la documentación y auditoría de los despliegues, complicando el cumplimiento de normativas y estándares de la industria. La trazabilidad y la capacidad de reproducir despliegues pasados se ven comprometidas, lo que puede tener implicaciones legales y de calidad.

4. Formulación del Problema de Investigación

¿Qué prácticas y estrategias se deben aplicar para optimizar la eficiencia, calidad y consistencia del proceso en la etapa de despliegue, teniendo en cuenta el enfoque Devops?

5. Objetivos

5.1. Objetivo General

Determinar un conjunto de prácticas y estrategias para el despliegue efectivo de microservicios, integrando prácticas de DevOps, con el fin de mejorar la eficiencia, la calidad y la consistencia del ciclo de vida del desarrollo de software.

5.2. Objetivos Específicos

- Realizar una revisión bibliográfica sobre teorías, buenas prácticas y procesos en el ámbito de DevOps para el despliegue automatizado de microservicios.
- Seleccionar y aplicar los instrumentos de medición para determinar las mejores Herramientas para el despliegue continuo con aplicaciones de microservicios.
- Seleccionar y aplicar los instrumentos para determinar la factibilidad de iniciar un proyecto de esta naturaleza sin entrenamiento previo en DevOps.
- Formular los pasos para comenzar a desarrollar una aplicación de microservicios desplegadas en un proceso DevOps, que mejore la eficiencia y calidad del ciclo de vida de desarrollo.

6. Diseño Metodológico

6.1. Tipo de la Investigación

El presente trabajo de investigación es de tipo descriptivo porque si bien se describe el objeto de estudio, los hechos y se acerca al problema, a su vez se intenta encontrar las causas del mismo y sobre ello construir una propuesta.

6.2 Métodos

6.2.1. Métodos Teóricos

Análisis documental: Revisión de literatura científica, artículos académicos, informes técnicos, manuales de usuario y documentación oficial de OpenProject.

Análisis – Síntesis: Para García et al. (2005), el análisis es una operación intelectual que posibilita descomponer mentalmente un todo complejo en sus partes y cualidades. El análisis permite la división mental del todo en sus múltiples relaciones y componentes. La síntesis es la operación inversa, que establece mentalmente la unión entre las partes, previamente analizadas y posibilita descubrir relaciones y características generales entre los elementos de la realidad.. El análisis se aplicó en la delimitación del problema, redacción del planteamiento, objetivos, elaboración del marco teórico, en la elaboración de los instrumentos de recolección de información. Síntesis, se aplicó en la formulación del problema de investigación y objetivos, así como en la estructura ración del marco teórico, aplicación de técnicas y diseño de instrumentos (encuesta y entrevista) y presentación de resultados y conclusiones de la presente monografía.

Método Deductivo: Según Tamayo (2008), este método consiste en la totalidad de reglas y procesos, con cuya ayuda es posible deducir conclusiones finales a partir de unos enunciados supuestos Fue aplicado para la para la redacción y comprobación del planteamiento del problema de investigación, así como para la formulación del diagnóstico las conclusiones y recomendaciones.

6.3. Técnicas

Análisis documental: Con esta técnica se pretende recopilar la teoría, procesos y filosofías de sistemas distribuidos como lo son los microservcios y los procesos DevOps.

Estudio de Caso: Se analizará plataforma CI/CD.

Observación: Consiste en observar y registrar los fenómenos tal como se presentan en su entorno natural, sin intervenir ni alterar las condiciones. La observación realizada es cualitativa.

6.4. Instrumentos

Fichas del análisis documental: Ayudará en la recolección y comprobación procesos dentro de las plataformas de la nube que ofrecen servicios para la automatización del despliegue.

Diario de Campo: Ayudará a presentar todos los datos recolectados en la integración de una aplicación de microservicios con la automatización del despliegue.

CAPITULO I: MARCO TEÓRICO

1.1. Marco Conceptual

1.1.1. Fundamentos de DevOps

DevOps es una metodología de desarrollo de software que integra los equipos de desarrollo y operaciones para mejorar la colaboración, la comunicación y la entrega rápida de software de alta calidad. En lugar de trabajar en silos separados, los equipos de DevOps trabajan juntos en todo el ciclo de vida del software, desde el desarrollo hasta la implementación y la monitorización.

Principios en los que se basa:

- DevOps promueve una cultura de colaboración y trabajo en equipo entre los desarrolladores de software y los profesionales de operaciones de TI. Esto implica derribar las barreras tradicionales entre los silos organizativos y fomentar la comunicación abierta y la colaboración en todas las etapas del ciclo de vida del desarrollo de software.
- DevOps se basa en la automatización de procesos para mejorar la eficiencia y la consistencia en el desarrollo, la prueba, la implementación y la operación de software. Esto incluye la automatización de la construcción y despliegue de aplicaciones, así como la gestión de configuraciones, pruebas y monitorización.
- DevOps promueve la entrega continua (Continuous Delivery) de software, lo que implica la capacidad de entregar cambios en el código de manera rápida, confiable y frecuente. Esto se logra a través de la automatización de procesos de integración y entrega continua (CI/CD), que permiten la entrega rápida y confiable de nuevas características y correcciones de errores.
- DevOps promueve una cultura de mejora continua, en la que los equipos buscan constantemente formas de mejorar los procesos, herramientas y prácticas de desarrollo y operaciones. Esto implica la retroalimentación continua, la evaluación de resultados y la adopción de prácticas y herramientas que permitan una entrega de software más rápida, confiable y eficiente.

1.1.2. Beneficios de su implementación:

- Entrega más rápida de software: DevOps ayudan a las empresas a entregar software de forma más rápida y con mayor frecuencia.
- Mejor calidad del software: DevOps ayudan a mejorar la calidad del software al reducir errores y mejorar la seguridad.
- Mayor satisfacción del cliente: Los clientes estarán más satisfechos con el software si se entrega de forma rápida y con alta calidad.
- Reducción de costes: DevOps ayudan a reducir los costes de desarrollo y operaciones de software.
- Ciclo de vida de desarrollo: El ciclo de vida de DevOps es un proceso iterativo que se compone de las siguientes fases:
- Planificación: En esta fase, los equipos de Dev y Ops se reúnen para planificar el próximo lanzamiento de software.
- Desarrollo: En esta fase, los desarrolladores escriben y prueban el código.
- Integración: En esta fase, el código se integra en la base de código principal.
- Pruebas: En esta fase, el software se prueba para detectar errores.
- Implementación: En esta fase, el software se implementa en producción.
- Monitorización: En esta fase, el software se monitoriza para detectar problemas y realizar mejoras.

1.1.3. Herramientas de CI/CD:

- Jenkins: Jenkins es una herramienta de integración continua y entrega continua (CI/CD) de código abierto popular. Se utiliza para automatizar la construcción, las pruebas y el despliegue de Microservicios.
- GitLab CI/CD: GitLab CI/CD es una herramienta de CI/CD integrada en la plataforma de desarrollo de software GitLab. Ofrece una amplia gama de funciones para automatizar el despliegue de Microservicios.
- Travis CI: Travis CI es una herramienta de CI/CD basada en la nube que es popular entre los proyectos de código abierto. Ofrece una interfaz de usuario sencilla y una amplia gama de integraciones.

1.1.4. Orquestadores de contenedores:

- Kubernetes: Kubernetes es una plataforma de orquestación de contenedores de código abierto popular. Se utiliza para automatizar la implementación, el escalado y la gestión de aplicaciones contenedorizadas, como los Microservicios.
- Docker Swarm: Docker Swarm es una plataforma de orquestación de contenedores nativa de Docker. Es una opción más ligera que Kubernetes y es más fácil de configurar y utilizar.
- Amazon Elastic Container Service (ECS): Amazon ECS es una plataforma de orquestación de contenedores totalmente gestionada que se ofrece como parte de Amazon Web Services (AWS). Es una buena opción para las empresas que ya utilizan AWS.

1.1.5. Plataformas de gestión de configuración:

- Ansible: Ansible es una herramienta de gestión de configuración popular que utiliza un lenguaje de dominio específico (DSL) para definir la configuración de la infraestructura y las aplicaciones.
- Chef: Chef es otra herramienta de gestión de configuración popular que utiliza un lenguaje de dominio específico para definir la configuración de la infraestructura y las aplicaciones.
- Puppet: Puppet es una herramienta de gestión de configuración madura que utiliza un lenguaje de dominio específico para definir la configuración de la infraestructura y las aplicaciones.

1.1.6. Herramientas de monitorización:

- Prometheus: Prometheus es un sistema de monitorización y alerta de código abierto popular. Se utiliza para recopilar y analizar datos de rendimiento de microservicios.
- Grafana: Grafana es una plataforma de visualización de datos de código abierto popular. Se utiliza para crear paneles de control para visualizar datos de rendimiento de microservicios de Prometheus y otras fuentes.

- Datadog: Datadog es una plataforma de monitorización y análisis basada en la nube. Ofrece una amplia gama de funciones para monitorizar el rendimiento, la seguridad y los registros de los Microservicios.

1.1.7. Herramientas de registro:

- ELK Stack:ELK Stack es una suite de código abierto popular para el registro y el análisis de datos. Está compuesto por Elasticsearch, Logstash y Kibana.
- Splunk: Splunk es una plataforma de registro y análisis comercial popular. Ofrece una amplia gama de funciones para recopilar, analizar y visualizar datos de registro.

1.1.8. La automatización y la eficiencia de la cultura DevOps

La automatización de procesos tiene un impacto significativo en la eficiencia operativa, la reducción de errores y el aumento de la productividad en diversos contextos, incluido el desarrollo de software. Aquí se detallan cómo se logran estos beneficios:

La automatización de procesos permite realizar tareas repetitivas de manera rápida y consistente, liberando tiempo y recursos humanos para actividades más estratégicas y de mayor valor añadido. En el desarrollo de software, esto se traduce en la automatización de actividades como la compilación, prueba y despliegue de código, así como la configuración y gestión de infraestructura. Al eliminar la necesidad de realizar estas tareas manualmente, se acelera el ciclo de desarrollo y se reduce el tiempo de entrega del software.

La automatización reduce la intervención humana en los procesos, lo que a su vez reduce la probabilidad de errores humanos. Los procesos automatizados siguen reglas y procedimientos predefinidos de manera consistente, lo que minimiza los errores causados por el cansancio, la distracción o la falta de atención. En el desarrollo de software, la automatización de pruebas de código, pruebas de integración y pruebas de regresión ayudan a identificar y corregir errores de manera más rápida y eficiente que los métodos manuales.

Al liberar a los equipos de desarrollo de tareas rutinarias y repetitivas, la automatización permite que se centren en actividades de mayor valor añadido, como el diseño de nuevas funcionalidades, la mejora de la experiencia del usuario y la optimización del rendimiento del

software. Además, la automatización permite a los equipos realizar más trabajo en menos tiempo, lo que aumenta la productividad general del equipo y la capacidad de entregar software de alta calidad de manera más rápida y eficiente.

La automatización de procesos en el desarrollo de software y otros contextos mejora significativamente la eficiencia operativa al acelerar los procesos, reducir los errores y aumentar la productividad del equipo. Al adoptar herramientas y prácticas de automatización, las organizaciones optimizan sus operaciones y ofrecer software de mayor calidad de manera más rápida y eficiente.

1.1.9. La entrega continua

La entrega continua de software es un enfoque que busca automatizar y mejorar el proceso de entrega de software de principio a fin, desde la integración de cambios en el código hasta su despliegue en producción. Este proceso se compone de dos prácticas principales: la integración continua (CI) y el despliegue continuo (CD). Aquí se analizan y explican ambas prácticas, así como la contribución de DevOps y la automatización a este proceso:

Integración continua (CI):

- La integración continua es una práctica en la que los cambios en el código se integran automáticamente en un repositorio compartido varias veces al día.
- Este proceso implica la automatización de la compilación, las pruebas y otras actividades de verificación de calidad para garantizar que los cambios se integren de manera suave y sin errores en el código existente.
- La CI permite a los equipos de desarrollo detectar y corregir problemas de manera temprana, facilitando la identificación rápida de errores y la entrega de software más estable y confiable.

Despliegue continuo (CD):

- El despliegue continuo es una extensión de la integración continua que automatiza el proceso de despliegue de software en entornos de producción.

- Con el despliegue continuo, los cambios que pasan las pruebas automatizadas en el entorno de CI se despliegan automáticamente en entornos de prueba y producción de manera continua y sin intervención manual.
- Esta práctica permite a los equipos de desarrollo entregar software de manera rápida y frecuente, lo que acelera el ciclo de retroalimentación y mejora la capacidad de respuesta a las necesidades del negocio y los usuarios finales.

1.1.10. Los Microservicios

Los Microservicios fueron creados para superar los desafíos de la arquitectura monolítica que dominaron inicialmente en el mercado y que también le permiten implementar servicios independientes.

Una arquitectura monolítica se puede decir que es similar a un contenedor grande en el que todos los componentes de software de una aplicación se ensamblan y empaquetan herméticamente.

Pero la arquitectura monolítica tiene varios desafíos:

- Las aplicaciones monolíticas no se pueden construir con diferentes tecnologías.
- No es confiable, incluso si una característica del sistema no funciona, entonces todo el sistema no funciona.
- Las aplicaciones monolíticas no se pueden escalar fácilmente, ya que cada vez que la aplicación necesita ser actualizada, el sistema completo tiene que ser reconstruido.
- Muchas características de las aplicaciones no se pueden crear e implementar al mismo tiempo.
- El desarrollo en aplicaciones monolíticas lleva mucho tiempo construirse, ya que todas y cada una de las características tienen que construirse una tras otra.
- Las características de las aplicaciones complejas tienen dependencias estrechamente acopladas.
- Estos desafíos fueron las principales razones que llevaron a la evolución de los Microservicios.

- Para tener una idea de Microservicios, veamos como una aplicación monolítica que se descompone en pequeñas microaplicaciones que se empaquetan y despliegan de forma independiente.
- Los Microservicios son un estilo arquitectónico que estructura una aplicación como una colección de pequeños servicios autónomos, modelados en torno a un dominio empresarial.

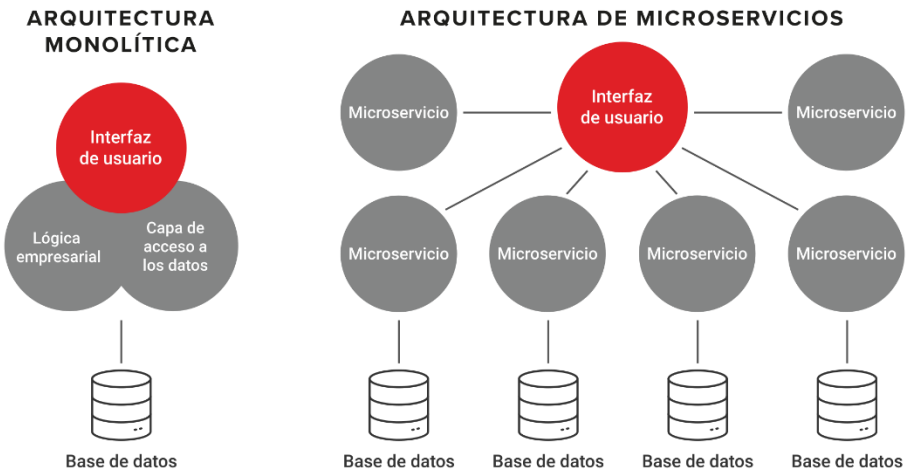


***Figura 1**. Comparación entre la arquitectura monolítica y la arquitectura de microservicios*

***Fuente:** CHAKRAY*

mos ver la diferencia entre las dos arquitecturas. La principal diferencia que observamos en el diagrama anterior es que todas las características inicialmente estaban bajo una sola instancia que compartía una sola base de datos. Pero luego, con los Microservicios, a cada característica se le asignó un Microservicio diferente, manejando sus propios datos y realizando diferentes funcionalidades.

En una arquitectura monolítica, toda la aplicación se desarrolla y despliega como una sola unidad monolítica, mientras que en la arquitectura de Microservicios, la aplicación se divide en componentes pequeños e independientes, cada uno enfocado en una función específica.

En una arquitectura monolítica, los diferentes módulos de la aplicación suelen estar fuertemente acoplados, lo que dificulta la modificación y el mantenimiento del código. En

cambio, en la arquitectura de Microservicios, son independientes y tienen acoplamiento mínimo, lo que facilita la evolución y la escalabilidad del sistema.

En una arquitectura monolítica, la escalabilidad suele ser limitada, ya que toda la aplicación debe escalarse en conjunto. En la arquitectura de Microservicios, éstos pueden escalarse de forma independiente según la demanda, lo que permite una asignación más eficiente de recursos.

En una arquitectura monolítica, toda la aplicación suele estar escrita en un solo lenguaje de programación y utilizar la misma tecnología. En la arquitectura de Microservicios, cada microservicio se desarrolla utilizando la tecnología y el lenguaje más adecuados para su función específica.

Las siguientes son algunas prácticas que se deben seguir al desarrollar Microservicios.

- Como desarrollador, cuando decida crear una aplicación, separe los dominios y sea claro con las funcionalidades. Utilice DDD para realizar esta separación.
- Cada Microservicio que diseñe se concentrará solo en un servicio de la aplicación.
- Asegúrese de haber diseñado la aplicación de tal manera que cada servicio se pueda implementar individualmente.
- Asegúrese de que la comunicación entre los Microservicios se realiza a través de un servidor sin estado(Stateless).
- Cada servicio puede ser refactorizado en servicios más pequeños, teniendo sus propios Microservicios.

1.1.11. Arquitectura de Microservicios

Los Microservicios son un estilo arquitectónico que estructura una aplicación como una colección de pequeños servicios autónomos, modelados en torno a un dominio empresarial.

Un sistema desarrollado con una arquitectura de Microservicios debe tener una estructura formada por diferentes componentes. Los componentes típicos de una Arquitectura de Microservicios son:

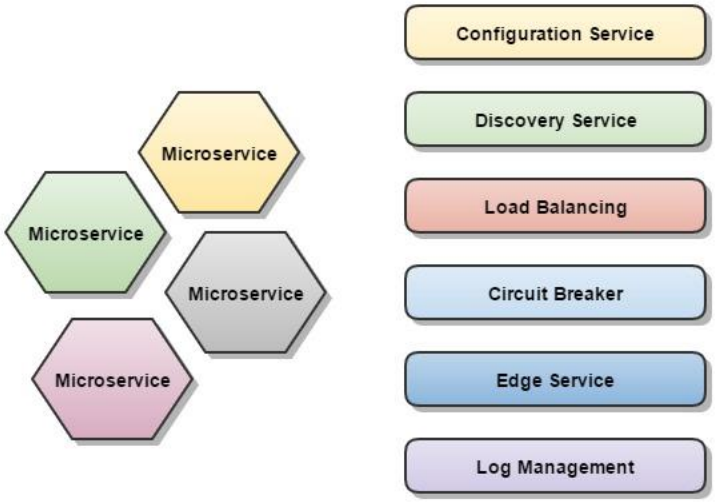


Figura 2. *Modelo de referencia*

Fuente: *Daniel Sanchez (2016)*

- Servidor de configuración central: Este componente se encargará de centralizar y proveer remotamente la configuración a cada Microservicio. Esta configuración se mantiene convencionalmente en un repositorio Git, lo que nos permitirá gestionar su propio ciclo de vida y versionado.
- Servicio de registro / descubrimiento: Este servicio centralizado será el encargado de proveer los endpoints de los servicios para su consumo. Todo Microservicio se registrará automáticamente en él en tiempo de bootstrap.
- Balanceo de carga (Load balancer): Este patrón de implementación permite el balanceo entre distintas instancias de forma transparente a la hora de consumir un servicio.
- Tolerancia a fallos (Circuit breaker): Mediante este patrón conseguiremos que cuando se produzca un fallo, este no se propague en cascada por todo el pipe de llamadas, y poder gestionar el error de forma controlada a nivel local del servicio donde se produjo.
- Servidor perimetral / exposición de servicios (Edge server): Gateway en el que se expondrán los servicios a consumir.
- Centralización de logs: Se hace necesario un mecanismo para centralizar la gestión de logs. Pues sería inviable la consulta de cada log individual de cada uno de los

Microservicios. Adicionalmente, también son interesantes los dos siguientes componentes.

- Servidor de Autorización: Para implementar la capa de seguridad (recomendable en la capa de servicios API).
- Monitorización: Mecanismos y algún dashboard para monitorizar aspectos de los nodos como, salud, carga de trabajo.

1.1.12. Características de los Microservicios

Figura 3. *Características de los microservicios.*

Fuente: *NucleusIT (2021)*

- Desacoplamiento: Los servicios dentro de un sistema están en gran medida desacoplados. Así que la aplicación en su conjunto se puede construir, alterar y escalar fácilmente. Los Microservicios son servicios independientes que se desarrollan, implementan y escalan de forma independiente.
- Componentización: Los Microservicios se tratan como componentes independientes que se pueden reemplazar y actualizar fácilmente.

- Capacidades empresariales: Los Microservicios son muy simples y se centran en una sola capacidad. Se alinean con las capacidades empresariales y desarrollan y se mantienen por equipos pequeños.
- Autonomía: Los desarrolladores y los equipos pueden trabajar de forma independiente, aumentando así la velocidad.
- Entrega continua: Permite lanzamientos frecuentes de software, a través de la automatización sistemática de la creación, prueba y aprobación de software.
- Responsabilidad: Los Microservicios no se centran en las aplicaciones como proyectos. En su lugar, tratan las aplicaciones como productos de los que son responsables
- Gobernanza descentralizada: El enfoque está en el uso de la herramienta adecuada para el trabajo adecuado. Eso significa que no hay un patrón estandarizado ni ningún patrón tecnológico. Los desarrolladores tienen la libertad de elegir las mejores herramientas útiles para resolver sus problemas.
- Agilidad: Cualquier nueva característica se puede desarrollar rápidamente y descartar de nuevo.
- Comunicación a través de APIs. Los Microservicios se comunican entre sí a través de interfaces bien definidas, como APIs RESTful.
- Tecnologías heterogéneas. Los Microservicios se pueden desarrollar utilizando diferentes lenguajes de programación, marcos y bases de datos.
- Escalabilidad horizontal. Los Microservicios se escalan horizontalmente agregando más instancias del servicio.

1.1.13. DevOps y Microservicios

La arquitectura de Microservicios y los principios de DevOps se complementan y refuerzan mutuamente en el desarrollo y despliegue de aplicaciones. Aquí se analiza y explica cómo la arquitectura de Microservicios se integra con los principios de DevOps y cómo la automatización afecta la implementación y el despliegue de esta arquitectura.

Desarrollo ágil y entrega continua:

- Los Microservicios permiten la descomposición de una aplicación en componentes independientes y autónomos. Esto facilita la adopción de metodologías ágiles de

desarrollo de software, ya que cada equipo trabaja de manera independiente en un servicio específico.

- La entrega continua es un principio fundamental de DevOps que busca acortar el ciclo de desarrollo y despliegue. Con la arquitectura de Microservicios, los equipos adoptan prácticas de entrega continua, como la integración continua y el despliegue continuo, para entregar cambios en los servicios de manera rápida y frecuente.

Escalabilidad y disponibilidad:

- Los Microservicios permiten escalar y desplegar servicios de forma independiente según la demanda, lo que mejora la escalabilidad y la disponibilidad de la aplicación en su conjunto. Esto se alinea con el principio de DevOps de escalabilidad, ya que los equipos gestionan de manera eficiente los recursos y la infraestructura para adaptarse a cambios en la carga de trabajo.
- La automatización juega un papel clave en la gestión de la escalabilidad en un entorno de microservicios. Los orquestadores de contenedores, como Kubernetes, permiten la automatización del escalado horizontal y vertical de los servicios en función de métricas de rendimiento y carga.

Retroalimentación rápida:

- La arquitectura de Microservicios facilita la entrega continua y el despliegue continuo de cambios en la aplicación, lo que acelera el ciclo de retroalimentación. Los equipos detectan y corrigen errores de forma más rápida y eficiente, lo que contribuye a la mejora continua de la aplicación.
- La automatización de pruebas, integración continua y despliegue continuo es fundamental para garantizar una retroalimentación rápida. Los equipos pueden automatizar pruebas unitarias, de integración y de aceptación utilizando herramientas de CI/CD, lo que permite una detección temprana de errores y una entrega más confiable de los cambios.

Automatización del despliegue:

- La automatización es esencial para gestionar y orquestar el despliegue de múltiples servicios en un entorno de Microservicios. Los equipos utilizan herramientas de CI/CD para automatizar la construcción, prueba, implementación y monitoreo de servicios de manera consistente y fiable.
- Los orquestadores de contenedores, como Docker Swarm o Kubernetes, permiten la automatización del despliegue y la gestión de contenedores en entornos de producción. Esto simplifica y agiliza el proceso de implementación y reduce la posibilidad de errores humanos.
- La arquitectura de Microservicios se integra con los principios de DevOps al facilitar el desarrollo ágil, la entrega continua y la retroalimentación rápida. La automatización es fundamental para la implementación exitosa de esta arquitectura, ya que permite gestionar de manera eficiente y fiable el desarrollo, prueba, despliegue y operación de servicios independientes.

1.2. Marco Teórico

1.2.1. Sistemas Complejos

La teoría de sistemas proporciona un marco conceptual poderoso para analizar la complejidad y comprender cómo funcionan los sistemas en diferentes contextos. Esta teoría nos ayuda a identificar diversos patrones, relaciones y procesos comunes que subyacen a una amplia variedad de fenómenos y sistemas, lo que permite abordar problemas de manera más holística y sistemática. Esta teoría considera que los sistemas están formados por componentes interrelacionados que trabajan juntos para cumplir un propósito o función específica.

A continuación, una ampliación del concepto de teoría de sistemas:

Un sistema está compuesto por partes o elementos que están interconectados y que interactúan entre sí. Estos componentes pueden ser físicos, como partes mecánicas de una máquina, o abstractos, como funciones en un programa de software.

Los componentes de un sistema están interconectados a través de relaciones y procesos que permiten la transferencia de información, energía o materia entre ellos. Estas interacciones son de diferentes tipos, como entradas y salidas, retroalimentación, o dependencias entre componentes.

Los sistemas se organizan en diferentes niveles jerárquicos, desde componentes individuales hasta sistemas más grandes que contienen sub-sistemas. La estructura de un sistema determina cómo se organizan y relacionan sus componentes para cumplir con sus objetivos.

Cada sistema tiene un propósito o función específica que determina su diseño y operación. Esta función es explícita, como la producción de bienes en una fábrica, o implícita, como la regulación del clima por parte de un ecosistema.

Los sistemas están interconectados y son interdependientes, lo que significa que los cambios en un componente pueden afectar a otros componentes del sistema. La retroalimentación es un mecanismo clave en los sistemas que permite ajustar y controlar su comportamiento en respuesta a cambios en el entorno o en el sistema mismo.

Los sistemas se adaptan y evolucionar con el tiempo para enfrentar cambios en su entorno o en sus requisitos. Esta capacidad de adaptación es fundamental para la supervivencia y el éxito a largo plazo de un sistema.

Al analizar la integración de prácticas de DevOps y la automatización en el ciclo de vida del desarrollo de software, se está utilizando la teoría de sistemas como un marco conceptual para comprender cómo interactúan y se relacionan los diferentes componentes y procesos involucrados en el desarrollo de software dentro del contexto de DevOps y la automatización.

Aquí se observa cómo se aplica la teoría de sistemas en este contexto:

En el desarrollo de software dentro de un entorno de DevOps, los diferentes componentes del sistema incluyen el código fuente, los sistemas de control de versiones, los entornos de desarrollo, pruebas y producción, las herramientas de automatización de CI/CD, los servidores de aplicaciones, entre otros. La teoría de sistemas ayuda a identificar y comprender cómo estos componentes están interconectados y cómo interactúan entre sí.

La teoría de sistemas permite analizar las interacciones y dependencias entre los diferentes componentes del sistema. Por ejemplo, cómo los cambios en el código fuente afectan el proceso de construcción y despliegue, o cómo las pruebas automatizadas proporcionan

retroalimentación al equipo de desarrollo. Este análisis ayuda a comprender cómo las prácticas de DevOps y la automatización impactan en todo el ciclo de vida del desarrollo de software.

La retroalimentación es un aspecto fundamental de la teoría de sistemas, y es especialmente relevante en el contexto de DevOps y la automatización. La retroalimentación proporcionada por las pruebas automatizadas, la monitorización del rendimiento y la retroalimentación del usuario final permite ajustar y mejorar continuamente el proceso de desarrollo de software. Además, la teoría de sistemas también aborda la adaptabilidad del sistema, es decir, cómo el sistema se ajusta y evolucionar en respuesta a cambios en el entorno o en los requisitos del negocio.

Proporciona herramientas y técnicas para visualizar y modelar sistemas complejos, lo que facilita la comprensión de su estructura y funcionamiento. Esto es especialmente útil cuando se trata de comprender la complejidad de los entornos de desarrollo de software modernos, que incluyen múltiples equipos, tecnologías y procesos interconectados.

Los sistemas complejos proporcionan un marco conceptual poderoso para analizar la integración de prácticas de DevOps y la automatización en el ciclo de vida del desarrollo de software, permitiendo comprender cómo los diferentes componentes y procesos interactúan entre sí.

1.3. Marco Contextual

La implementación de DevOps y la automatización del despliegue de microservicios se ven influenciadas por una serie de factores socioeconómicos, culturales e institucionales. A continuación, se detallan algunos de los más importantes:

En el entorno empresarial actual, existe una creciente presión para entregar software de forma rápida y fiable. Esto ha impulsado la adopción de DevOps y la automatización del despliegue de microservicios, ya que estas prácticas ayudan a las empresas a acortar los ciclos de desarrollo y entrega. Los costos de desarrollo de software han ido aumentando en los últimos años, lo que ha llevado a las empresas a buscar formas de mejorar la eficiencia. DevOps y la automatización del despliegue de microservicios ayudan a las empresas a reducir los costos de desarrollo de software al automatizar tareas manuales y reducir los errores. Existe una

escasez global de desarrolladores de software cualificados, lo que ha dificultado a las empresas encontrar el talento necesario para desarrollar y mantener aplicaciones de software. DevOps y la automatización del despliegue de microservicios ayudan a las empresas a superar esta escasez al permitir que los desarrolladores se centren en tareas de mayor valor y reduzcan el tiempo que dedican a tareas manuales repetitivas.

DevOps requiere una cultura de colaboración entre los equipos de desarrollo y operaciones. Esto puede ser un desafío en algunas organizaciones, donde hay silos entre los dos equipos. Sin embargo, es importante fomentar una cultura de colaboración para que DevOps tenga éxito. DevOps se basa en una mentalidad de mejora continua, lo que significa que los equipos deben buscar constantemente formas de mejorar sus procesos y herramientas. Esta mentalidad suele ser difícil de adoptar en algunas organizaciones, donde puede haber resistencia al cambio. Sin embargo, es importante adoptar una mentalidad de mejora continua para que DevOps tenga éxito a largo plazo. También DevOps requiere que las organizaciones estén abiertas a adoptar nuevas tecnologías y enfoques. Esto es un desafío en algunas organizaciones, donde existe una aversión al riesgo. Sin embargo, es importante estar abierto a nuevas tecnologías para que DevOps tenga éxito.

La implementación de DevOps y la automatización del despliegue de microservicios requiere el apoyo de la gerencia. La gerencia debe estar dispuesta a invertir en las herramientas y la capacitación necesarias, y también debe crear una cultura que respalde la colaboración y la mejora continua. Los empleados deben tener las habilidades y los conocimientos necesarios para implementar DevOps y la automatización del despliegue de Microservicios. Esto requiere capacitación y desarrollo adicional. Hay una serie de herramientas y tecnologías disponibles para ayudar a las organizaciones a implementar DevOps y la automatización del despliegue de Microservicios, por ello es importante seleccionar las herramientas y tecnologías adecuadas para las necesidades de la organización.

En general, el contexto socioeconómico, cultural e institucional juega un papel importante en la implementación exitosa de DevOps y la automatización del despliegue de Microservicios. Las organizaciones deben tener en cuenta estos factores al desarrollar e implementar sus estrategias de DevOps.

1.3.1. Beneficios potenciales de la automatización del despliegue de microservicios

Entrega más rápida de software: La automatización del despliegue ayuda a las empresas a entregar software de forma más rápida al reducir el tiempo necesario para desplegar nuevos cambios en producción.

Mayor fiabilidad y eficiencia: La automatización del despliegue de microservicios ayuda a mejorar la fiabilidad del software al reducir la posibilidad de errores humanos a causa de las tareas manuales repetitivas y propensas a errores. La automatización del despliegue ayuda a mejorar la eficiencia de las operaciones de TI al reducir la necesidad de intervención manual.

Consistencia: Al estandarizar y automatizar los procesos de despliegue, se garantiza una mayor consistencia en los entornos de desarrollo, pruebas y producción.

Mayor escalabilidad: La automatización del despliegue ayuda a escalar las implementaciones de Microservicios de forma más fácil al permitir que las empresas desplieguen nuevos servicios o escalen los servicios existentes de forma rápida y sencilla.

Retroalimentación rápida: La automatización del despliegue permite obtener una retroalimentación rápida sobre los cambios realizados, lo que facilita la detección temprana de problemas y la iteración continua.

Mejoras en la moral del equipo: La automatización del despliegue ayuda a mejorar la moral del equipo al liberar a los desarrolladores de tareas manuales repetitivas y permitirles centrarse en un trabajo más creativo y gratificante.

Desafíos y consideraciones:

- Complejidad operativa: La implementación de procesos automatizados introduce una mayor complejidad operativa, especialmente en entornos distribuidos y heterogéneos, ya que requiere una comprensión de las herramientas y tecnologías involucradas, así como de los procesos de negocio subyacentes.
- Seguridad: La automatización del despliegue requiere una atención especial a la seguridad, ya que los errores en la configuración automatizada exponen la

infraestructura y los datos a riesgos de seguridad. Debe diseñarse e implementarse de forma segura para evitar que los piratas informáticos o los usuarios malintencionados implementen código malicioso en producción

- Gestión del cambio: La automatización del despliegue implica cambios en los procesos y las herramientas utilizadas, lo que genera resistencia y requerir una gestión cuidadosa del cambio organizacional. Es importante gestionar el cambio de manera efectiva para minimizar la interrupción del negocio
- Costos: La implementación de una solución de automatización del despliegue de Microservicios puede ser costosa, especialmente si se requieren nuevas herramientas o tecnologías.
- Dependencias: Los Microservicios tienen dependencias entre sí, lo que complica el proceso de automatización del despliegue.
- Monitoreo y mantenimiento: Es necesario establecer un sólido sistema de monitoreo y mantenimiento para asegurar que los procesos automatizados funcionen correctamente y puedan ser ajustados y mejorados según sea necesario. En caso de ocurrir problemas identificar y resolverlos rápidamente.
- Cultura y colaboración: La automatización del despliegue requiere una cultura organizacional que fomente la colaboración entre equipos de desarrollo y operaciones, así como una mentalidad de mejora continua.

Consideraciones adicionales:

- Madurez de la organización: La automatización del despliegue de microservicios es más adecuada para organizaciones que ya tienen un cierto nivel de madurez en DevOps y prácticas de desarrollo de software.
- Cultura organizacional: La automatización del despliegue de microservicios requiere una cultura que apoye la innovación y la experimentación.

Habilidades del equipo: Las empresas deben tener los empleados con las habilidades y los conocimientos necesarios para implementar y mantener una solución de automatización del despliegue de microservicios.

1.3.2. La gestión del cambio organizacional

La gestión y adopción de cambios en procesos, estructuras y culturas organizacionales es un aspecto crucial para implementar exitosamente DevOps y la automatización en una empresa. A continuación, se analizan y explican cómo las organizaciones abordan estos cambios y cómo aplicar para comprender los desafíos y estrategias involucradas en la implementación de DevOps y la automatización:

Gestión del cambio organizacional:

- Las organizaciones enfrentan resistencia al cambio debido a la naturaleza disruptiva de las nuevas prácticas y tecnologías como DevOps y la automatización.
- Para gestionar el cambio de manera efectiva, las organizaciones necesitan comunicar claramente la visión y los beneficios de la adopción de DevOps y la automatización, involucrar a los empleados en el proceso de cambio, y proporcionar capacitación y apoyo adecuados para ayudar a los equipos a adaptarse a los nuevos procesos y herramientas.

Adaptación de procesos:

- La implementación de DevOps y la automatización requiere cambios en los procesos existentes para integrar nuevas prácticas como la integración continua, el despliegue continuo y la colaboración entre equipos.
- Las organizaciones necesitan evaluar y ajustar sus procesos de desarrollo, pruebas, despliegue y operaciones para aprovechar al máximo las nuevas prácticas y herramientas.

Reestructuración organizacional:

- La adopción de DevOps y la automatización requiere cambios en la estructura organizacional para fomentar la colaboración y la responsabilidad compartida entre equipos de desarrollo, operaciones y otros departamentos.

- Las organizaciones consideran la formación de equipos multifuncionales, la eliminación de silos organizativos y la implementación de estructuras de gobierno ágiles para facilitar la entrega continua de software.

Transformación cultural:

- La cultura organizacional desempeña un papel crucial en el éxito de la implementación de DevOps y la automatización. Las organizaciones necesitan fomentar una cultura de colaboración, transparencia, aprendizaje continuo y mejora.
- Esto implica cambios en las actitudes, comportamientos y valores de los empleados, así como en las prácticas de gestión y liderazgo.

Al comprender estos desafíos y estrategias, las organizaciones deben diseñar e implementar un enfoque de cambio organizacional que facilite la adopción exitosa de DevOps y la automatización. Esto incluye la gestión proactiva de la resistencia al cambio, la adaptación de procesos y estructuras organizacionales, y la promoción de una cultura que fomente la colaboración, la innovación y la mejora continua.

CAPITULO II: DIAGNÓSTICO

2.1. Introducción

Con la ayuda de la teoría de sistemas, para determinar las mejores prácticas y estrategias del despliegue efectivo de microservicios junto con prácticas DevOps, se planteó el siguiente escenario:

- Configurar un canal de despliegue e integración (pipeline CI/CD) en múltiples entornos para una aplicación de microservicios que se ejecuta en un cluster de Kubernetes
- Siempre que exista un nuevo commit, la aplicación debe implementarse o desplegarse automáticamente en el clúster con diferentes entornos.

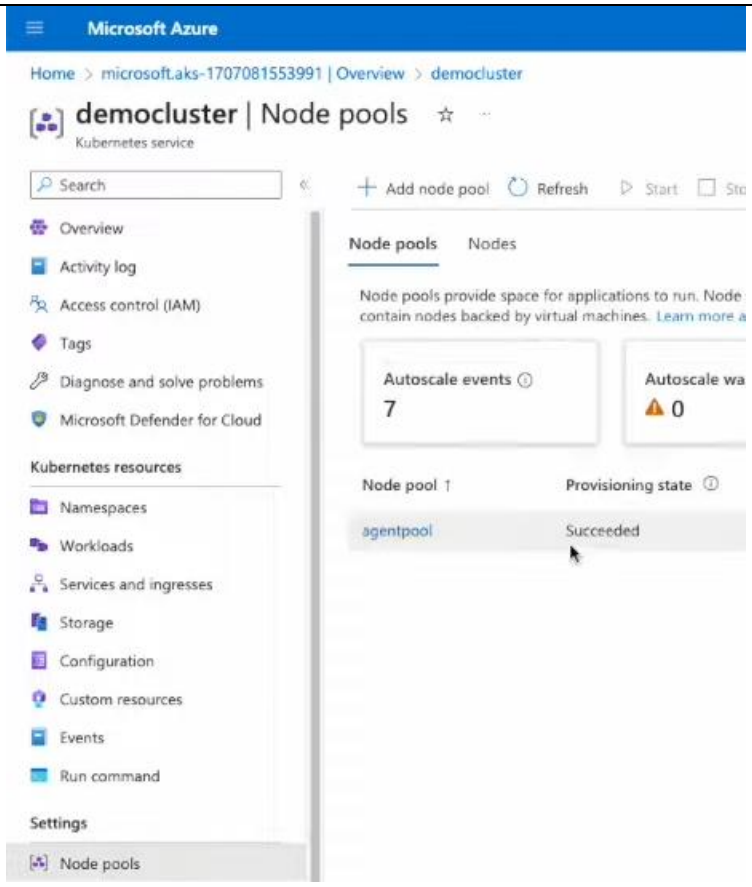
2.2. Presentación y Análisis de Resultados

2.2.1. Resultados del Análisis Documental

Como la plataforma de Microsoft Azure es bastante popular en la comunidad hispana, se optó por documentar la implementación básica de sus servicios, antes de poder realizar la integración de microservicios con la filosofía DevOps. El objetivo es llenar todos los prerrequisitos de la plataforma antes de realizar el objetivo mismo de la presente investigación. A continuación, se muestra cronológicamente la configuración:

Cuadro 1:Guía de Observación para la preparación de la infraestructura

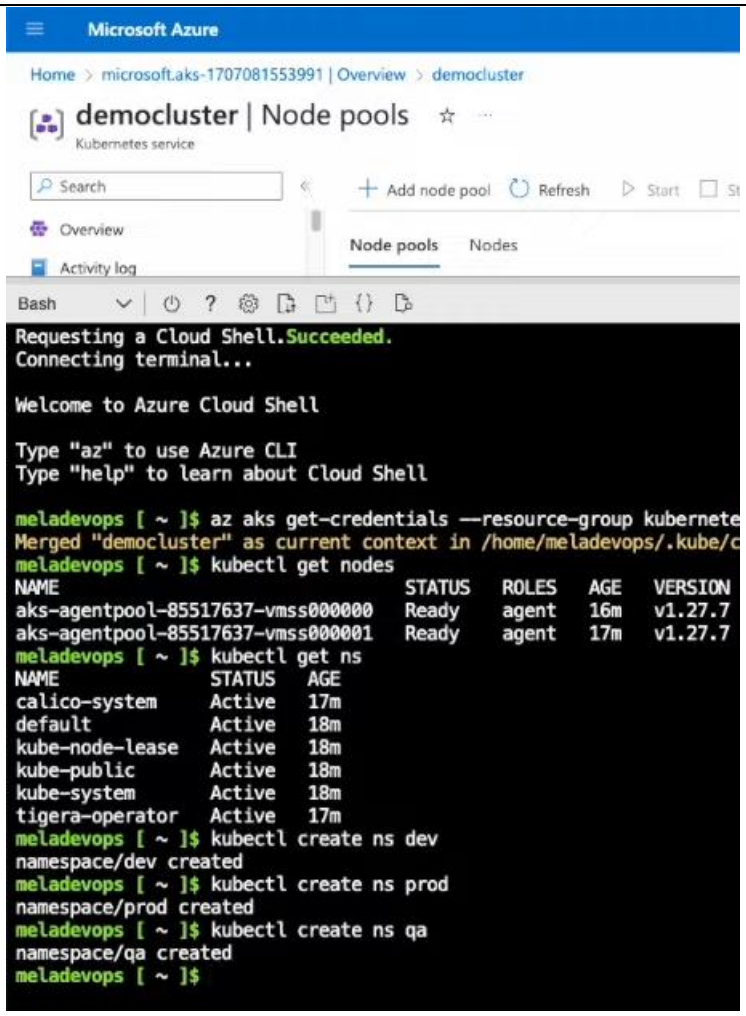
Paso 1	Creación de 2 clusters a través del servicio de Azure
Tarea	**Captura de pantalla de tarea**

Creación de clúster de AKS	

Fuente: Elaboración propia

Cuadro 2: Guía de Observación para la preparación de la infraestructura (Continuación)

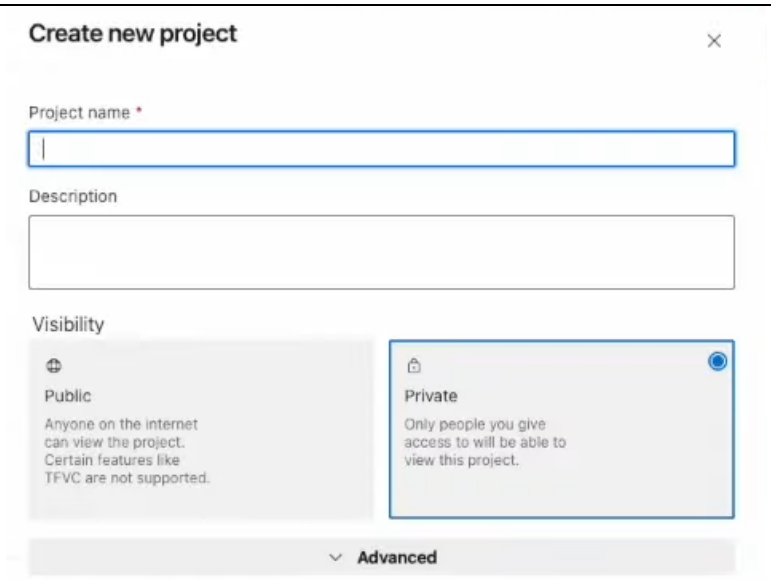
Paso 2	Creación de tres espacios de nombres para: Desarrollo (Dev), Aseguramiento de la Calidad (QA) y Producción (Prod)
Tarea	**Captura de pantalla de tarea**

Creación de espacios de nombres	

Fuente: Elaboración propia

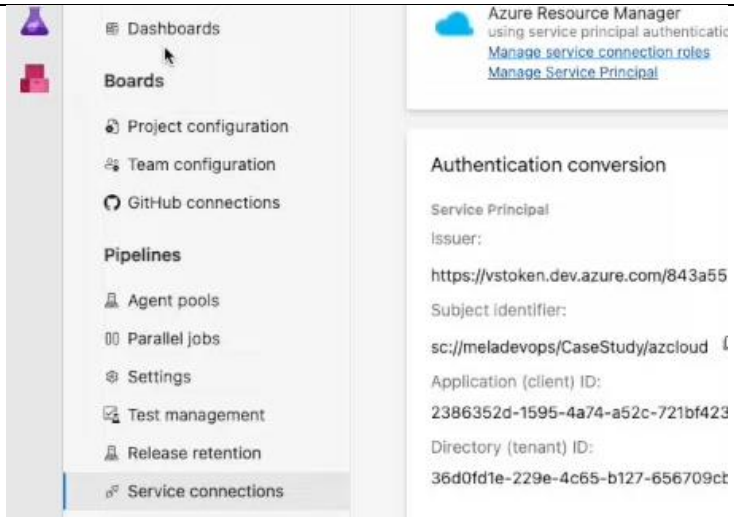
Cuadro 3:Guía de Observación para la compilación y el lanzamiento

Paso 3	En la consola gráfica de Azure se introduce el nombre y la descripción del nuevo proyecto DevOps
Tarea	**Captura de pantalla de tarea**

Creación de del proyecto Azure DevOps	

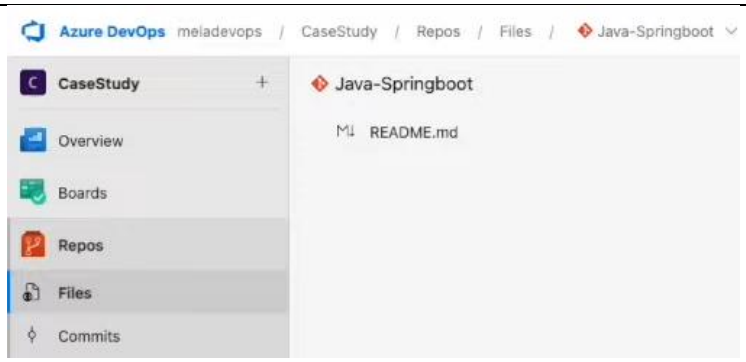
Fuente: Elaboración propia

Cuadro 4:Guía de Observación para la compilación y el lanzamiento (Continuación)

Paso 4	Se creó el servicio de conexión para la autenticación conexión con el clúster AKS
Tarea	**Captura de pantalla de tarea**
Creación de Servicio de Conexión	

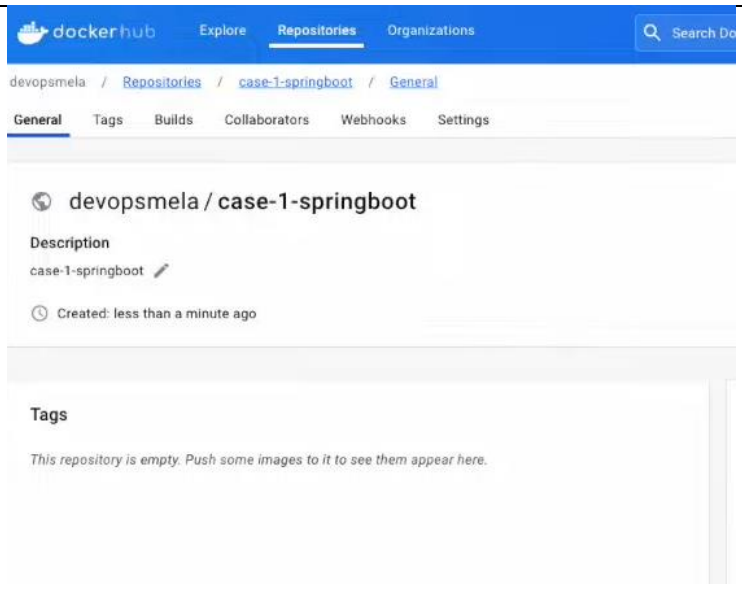
Fuente: Elaboración propia

Cuadro 5:Guía de Observación para la compilación y el lanzamiento (Continuación)

Paso 5	El repositorio está vacio, listo para utilizarse con Git
Tarea	**Captura de pantalla de tarea**
Creación de repositorios de Azure	

Fuente: Elaboración propia

Cuadro 6:Guía de Observación para la compilación y el lanzamiento (Continuación)

Paso 6	En la consola gráfica de Docker se crea el contenedor vinculado con el proyecto
Tarea	**Captura de pantalla de tarea**
Creación de repositorio de Imagen Docker	

Fuente: Elaboración propia

2.2.2. Resultados del Estudio de Caso: Springboot con AKS

Estando definida la plataforma en la que se realizará la implementación de una aplicación con microservicios y el despliegue en Kubernetes (DevOps). Se procedió a detallar las tecnologías a utilizar:

- Azure DevOps
- Git
- Kubernetes
- Docker

Se procedio a plantear el escenario y se determinó el siguiente enfoque:

- Crear un repositorio (GitHub) y proyectos de Azure DevOps
- Crear un canal de compilación y lanzamiento
- Cree un servicio principal para conectarse al clúster de AKS
- Configurar activadores de implementación e integración continua (CI/CD)
- Configurar variables de canal, si las hay

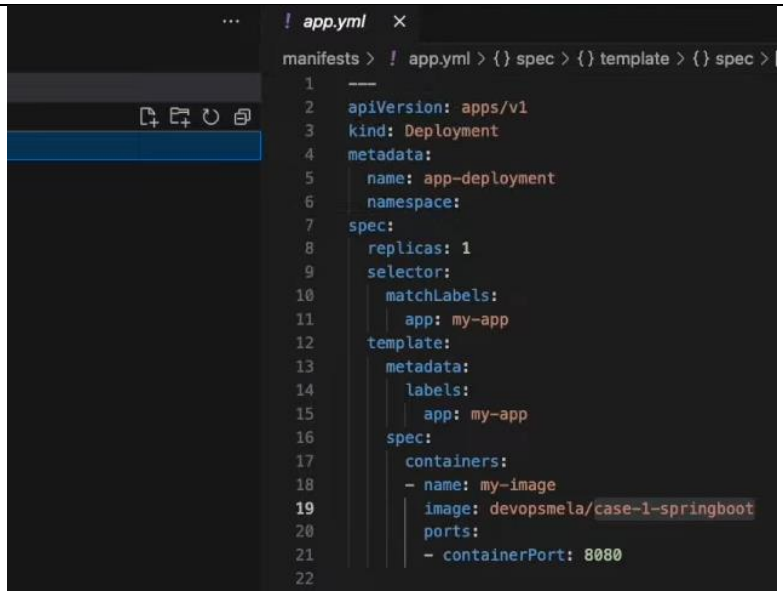
Y se detallaron las partes del diagnóstico:

- Una aplicación Java Springboot que se desplegarse en un clúster Azure Kubernetes Service (AKS), y
- El despliegue a tres entornos de AKS: Desarrollo, Aseguramiento de la Calidad (QA) y Producción, si existe un nuevo commit.

A continuación, se muestra la primera parte del diagnóstico utilizando un diario de campo, que registra la creación del canal CI/CD automatizada a través de archivos (manifests). Se describen los pasos más relevantes para el presente estudio:

Cuadro 7:Creación del Canal CI/CD

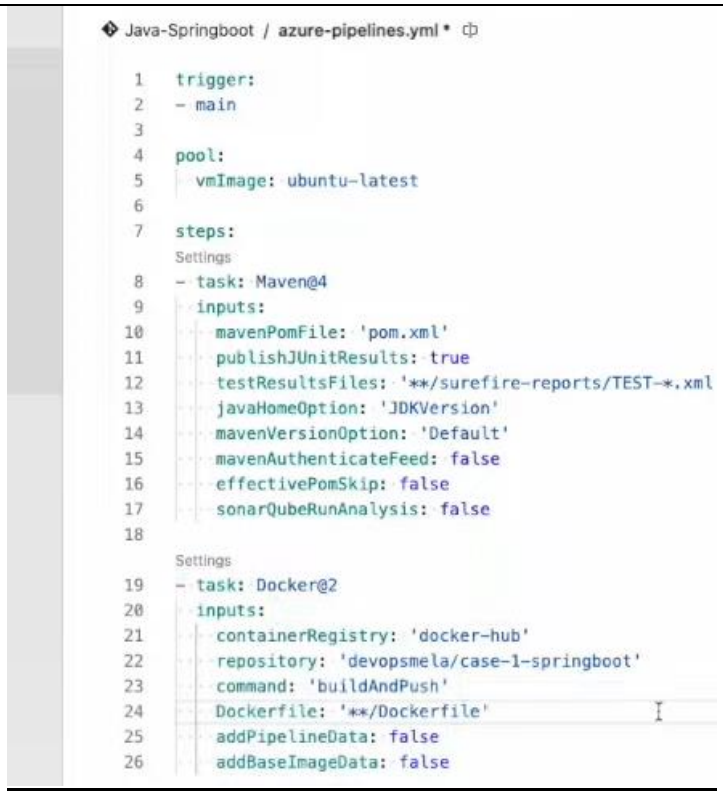
Diario de Campo	
Descripción	En el archivo se muestra: el nombre de la aplicación (app-deployment), y la ruta (devopsmela/case-1-springboot) de la cual

	el contenedor está sacando una imagen del repositorio (puerto del contenedor es 8080)
Unidades de análisis	Canal CI/CD
Tarea	**Captura de pantalla de tarea**
Archivo para despliegue	

Fuente: Elaboración propia

Cuadro 8:Creación del Canal CI/CD (Continuación)

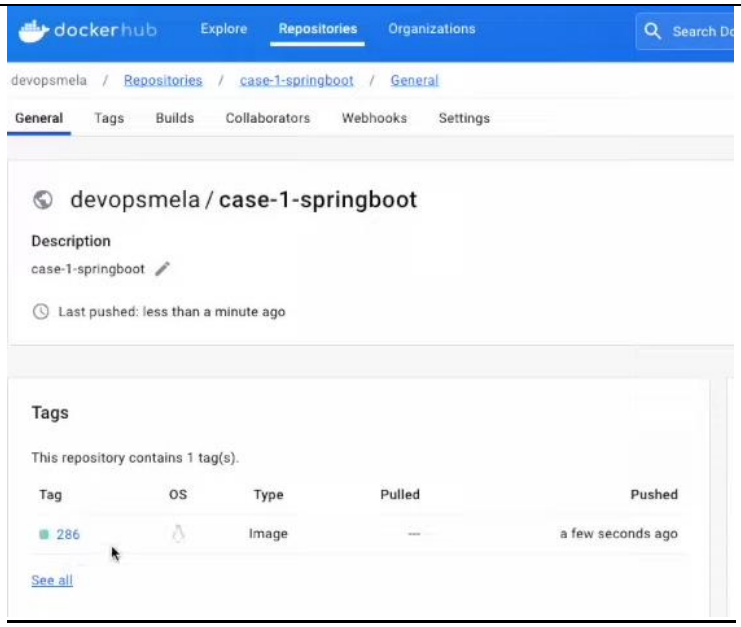
Diario de Campo	
Descripción	El archivo de configuración .yaml . En general se tiene dos tareas. En la primera se construye el proyecto (build y test) Maven, y en la segunda se envía el artefacto (build y push) el repositorio remoto (Docker)
Unidades de análisis	Canal CI/CD
Tarea	**Captura de pantalla de tarea**

Archivo para la creación de canal CI/CD	

Fuente: Elaboración propia

Cuadro 9: Creación del Canal CI/CD (Continuación)

Diario de Campo	
Descripción	El repositorio de Docker efectivamente contiene la nueva imagen creada
Unidades de análisis	Canal CI/CD
Tarea	**Captura de pantalla de tarea**

Canal CI/CD creado	

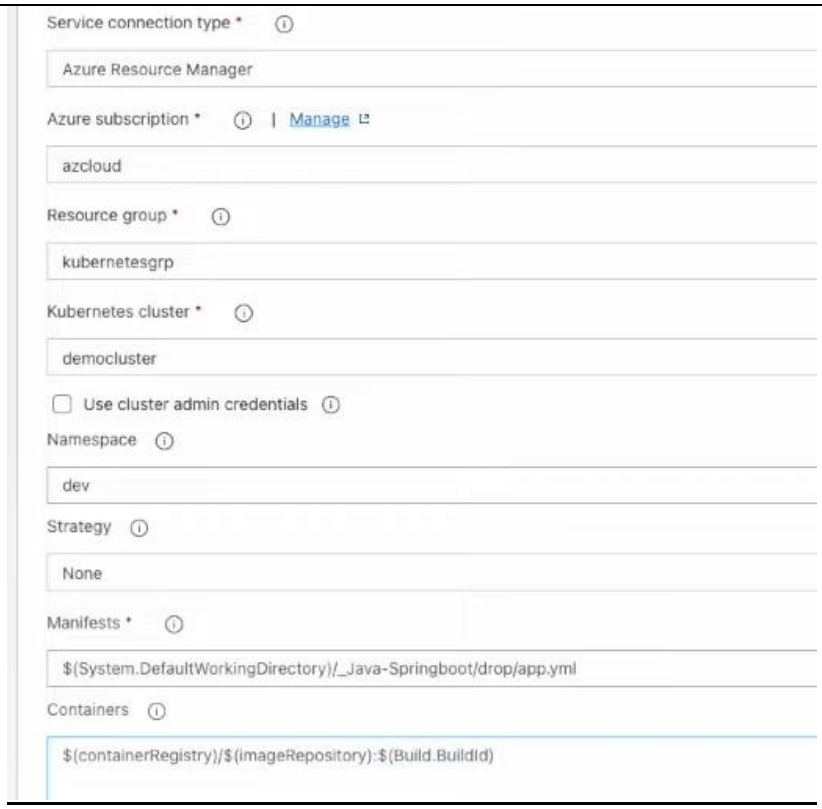
Fuente: Elaboración propia

Hasta este punto se ha cumplido con la primera parte de este diagnóstico, se ha creado un canal CI/CD para una aplicación de microservicios Java Springboot (con Maven en la ruta devopsmela/case-1-springboot) que se ejecuta en Azure Kubernetes Service (AKS).

Proseguimos con la segunda parte, que describe el despliegue de una aplicación basada en microservicios a tres diferentes instancias de Kubenetes:

Cuadro 10: Creación de despliegue

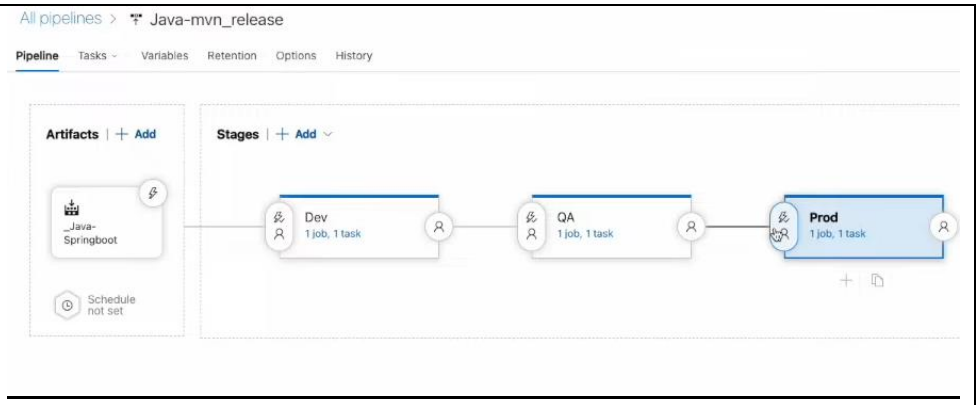
Diario de Campo	
Descripción	Configuración de etapa (stage) en la que se hace el despliegue a Kubernetes en base a las variables: ContainerRegistry. ImageRepositoy y BuildID. Las tres ayudarán a determinar cuándo ocurre un nuevo commit, para luego desplegar de forma automática en 3 entornos.
Unidades de análisis	Despliegue a clúster de Kubernetes
Tarea	**Captura de pantalla de tarea**

Determinación de la ocurrencia de un nuevo commit	

Fuente: Elaboración propia

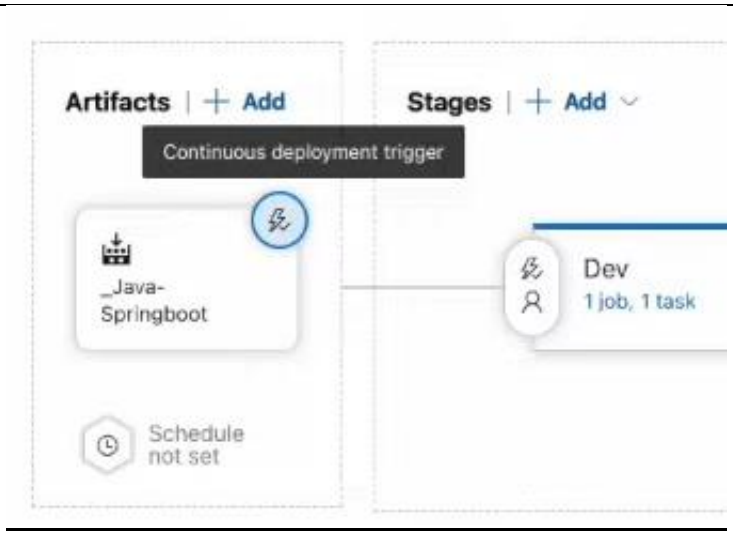
Cuadro 11: Creación de despliegue (Continuación)

Diario de Campo	
Descripción	El artefacto es creado a través de un archivo yaml. Las etapas (stage) para Dev, QA y Prod son creadas en el paso anterior (cuadro 11) y se repiten para los tres.
Unidades de análisis	Despliegue a clúster de Kubernetes
Tarea	**Captura de pantalla de tarea**

Despliegue creado	

Fuente: Elaboración propia

Cuadro 12:Creación de despliegue (Continuación)

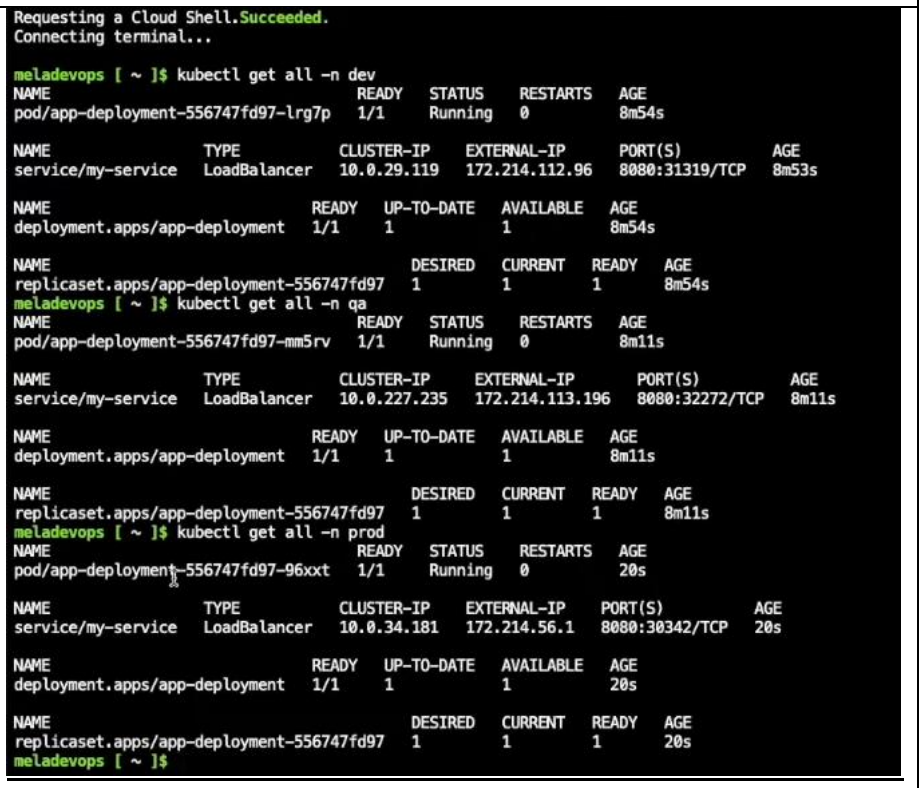
Diario de Campo	
Descripción	El artefacto es creado a través de un archivo yaml. Las etapas (stage) para Dev, QA y Prod son creadas en el paso anterior (cuadro 11) y se repiten para los tres.
Unidades de análisis	Despliegue a clúster de Kubernetes
Tarea	**Captura de pantalla de tarea**
Despliegue creado	

Fuente: Elaboración propia

El artefacto de Java está configurado para el despliegue continuo (ver cuadro 12), el activador es la ocurrencia de un nuevo commit. Cuando ocurre este, se inicia de forma automática el

proceso de construcción para los tres entornos de AKS: Desarrollo, Aseguramiento de la Calidad (QA) y Producción.

Cuadro 13:Despliegue a 3 entornos

Diario de Campo	
Descripción	El log que demuestra que los Kubernetes: `dev, qa y prod` han sido desplegados exitosamente sin ningún inconveneinte
Unidades de análisis	Despliegue a clúster de Kubernetes
Tarea	**Captura de pantalla de tarea**
Despliegue exitoso de los AKS	

Fuente: Elaboración propia

2.3. Conclusiones Generales del Diagnóstico

Para incorporar la automatización del despliegue en un sistema desarrollado bajo la arquitectura de microservicios, se debe realizar las siguientes consideraciones:

Primero: Determinar pruebas piloto de aplicaciones con microservicio sencillas y hacia pocos entornos:

- En lugar de compilar un abundante código inicialmente
    - primero tener una aplicación básica (Hola Mundo) y
    - luego introducir funciones del mundo real.
- Esto reduce en gran medida la carga de trabajo del desarrollador. Al mismo tiempo, se pueden automatizar (a través de archivos):
    - La creación de etapas (Stages en el caso de Azure), seguidas de
    - La creación del artefacto (Java en el presente caso).
- Sin embargo, un punto importante a considerar es la determinación de todas las *posibles variables* que logren iniciar un nuevo commit.

Segundo: En cuanto la frecuencia de los lanzamientos de las versiones de software, esta puede ser alta si los paquetes están listos para su lanzamiento y se han probado en un entorno idéntico al de producción. La secuencia sería entonces:

- El lanzamiento de la función/software a un subconjunto de usuarios, probar el software/función y luego
- Enviarlo a un conjunto más amplio de usuarios una vez que haya tenido éxito.

Tercero: En cuanto a la integración, se puede evitar una integración masiva justo antes del lanzamiento, siempre y cuando el equipo se comprometa a realizar commits diarios desde cada rama (branch) al tronco principal del repositorio, para ayudar a dedicar más tiempo al desarrollo y menos al control de versiones.

Conclusiones

- Se determinó que la mejor forma de desplegar aplicaciones de microservicios con procesos DevOps es utilizando plataformas de servicios como Azure o AWS que ya vienen con todo un ecosistema de herramientas y procesos de automatización, cuyo diseño delinea las mejores estrategias para un eficiente ciclo de vida de desarrollo. Estas plataformas proporcionan servicios gestionados para la orquestación de contenedores, monitoreo, escalado automático y gestión de configuraciones, entre otros.
- Utilizar estas plataformas permite a los desarrolladores enfocarse en el desarrollo de la lógica de negocio en lugar de preocuparse por la infraestructura subyacente. Además, estas plataformas siguen las mejores prácticas de la industria y ofrecen soporte robusto para la automatización y la integración continua (CI/CD), por lo que la adopción de estas plataformas contribuye a un ciclo de vida de desarrollo más eficiente y ágil, permitiendo una rápida iteración y despliegue de nuevas funcionalidades.
- Se realizó una revisión bibliográfica sobre los microservicios con DevOps que permitió identificar y analizar diversas prácticas y enfoques en la implementación de microservicios con DevOps, lo que proporcionó una base sólida para la selección de las mejores estrategias y herramientas.
- Se pudieron establecer métricas clave y métodos de diagnóstico que son cruciales para evaluar el rendimiento, la estabilidad y la eficiencia de los microservicios. Estas métricas incluyen tiempo de respuesta, uso de recursos, tasas de error, y otros indicadores de salud del sistema.
- Se determinó que cualquier herramienta para el canal CI/CD es eficiente para el despliegue continuo con aplicaciones de microservicios, porque se realizaron las pruebas en una plataforma que no es popular entre los desarrolladores.

- La investigación y pruebas realizadas confirmaron que, con las herramientas y prácticas adecuadas, un solo desarrollador puede manejar el despliegue de aplicaciones de microservicios en sistemas distribuidos. Esto se debe a la alta automatización y la simplificación de procesos que ofrecen las plataformas modernas. La utilización de contenedores y orquestadores como Docker y Kubernetes facilita este proceso, permitiendo una implementación consistente y reproducible de los microservicios.
- Se formuló los pasos necesarios para comenzar a desarrollar una aplicación de microservicios, incluso cuando el equipo tiene cero experiencia en DevOps, logrando la automatización del despliegue en cada commit, mediante la integración de herramientas de CI/CD que automatizan la construcción, prueba y despliegue de los microservicios. Esto asegura que cada cambio en el código se despliegue de manera consistente y sin intervención manual.

Recomendaciones

- Utilizar los instrumentos de medición del presente estudio con una aplicación cuyas funciones sean atractivas al mercado comercial. Esto asegura que el tiempo y los recursos invertidos en el desarrollo y despliegue de microservicios se traduzcan en un valor real y tangible para los usuarios finales.
- Los instrumentos de medición deben incluir métricas de rendimiento, escalabilidad, tiempo de respuesta, uso de recursos y satisfacción del usuario, entre otros. Estos indicadores ayudarán a evaluar la efectividad de los microservicios y a realizar ajustes necesarios para optimizar su funcionamiento.
- La automatización es un componente clave en cualquier estrategia de DevOps, y debe abarcar tanto las pruebas así como el monitoreo continuo de los microservicios, por ello se recomienda incluir en el proceso de automatización las pruebas y el monitoreo.
- Las pruebas automatizadas deben incluir pruebas unitarias, pruebas de integración, pruebas de rendimiento y pruebas de seguridad. Esto asegura que los microservicios funcionen correctamente en todos los escenarios posibles y que cualquier problema sea detectado y corregido antes del despliegue en producción.
- El monitoreo continuo es esencial para mantener la salud del sistema y garantizar un rendimiento óptimo. Herramientas como Prometheus, Grafana, y ELK Stack (Elasticsearch, Logstash, Kibana) pueden ser utilizadas para recopilar, analizar y visualizar datos en tiempo real.
- Implementar alertas automatizadas que permita a los equipos de desarrollo y operaciones responder rápidamente a cualquier anomalía o degradación en el servicio, minimizando el impacto en los usuarios finales.
- Conformar un equipo para el desarrollo de aplicaciones de microservicios que pueda migrar a la cultura DevOps. La transición a una cultura DevOps requiere la formación

de un equipo multifuncional que incluya desarrolladores, ingenieros de operaciones, expertos en seguridad y profesionales de calidad. Este equipo debe estar capacitado en prácticas y herramientas de DevOps, como la integración continua (CI), la entrega continua (CD), la infraestructura como código (IaC) y la gestión de configuraciones.

- Es crucial fomentar una mentalidad colaborativa donde todos los miembros del equipo compartan la responsabilidad del desarrollo y la operación de los microservicios. Esto implica la eliminación de silos y la promoción de una comunicación abierta y constante.
- La capacitación continua y el desarrollo profesional son importantes para mantenerse al día con las últimas tecnologías y mejores prácticas en el ámbito de DevOps y microservicios. Inversiones en talleres, conferencias y cursos especializados pueden ser altamente beneficiosas.
- La adopción de metodologías ágiles también puede complementar la cultura DevOps, permitiendo iteraciones rápidas, feedback constante y una mejora continua en el desarrollo y despliegue de aplicaciones.

Referencias bibliográficas

Morles, Victor(2005). Educación de posgrado o educación avanzada en Venezuela: ¿Para qué?. Universidad Pedagógica Experimental Libertador. Caracas Venezuela.

Moreno, Zully(2004). Diagnóstico y perspectiva de los estudios de posgrado en Bolivia. IESALC-UNESCO. Consultado: [20 Diciembre 2021]. Disponible en: [https://docplayer.es/16919161-Diagnostico-y-perspectiva-de-los-estudios-de-postgrado-en-bolivia.html]

Simón lazaro, Mariano(2008). Aprendizaje electrónico móvil o Mobile Learning. [En linea]. Consultado: [20 diciembre 2021]. Disponible en: [http://m-aprendizaje.blogspot.com/2008/12/aprendizaje-electrnico-mvil-o-mobile.html]

Blancarte Iturralde, Oscar Javier(2020), Introducción a la arquitectura de software: un enfoque práctico. Oscar Blancarte Blog Software Architect.

Shvets, Alexander(2019). Sumérgete en los patrones de diseño. Libro electrónico.

Sommerville, Ian(2011). Ingeniería de Software(9na. Edición). Pearson Educación de México.

Suaznabar Claros, Fernando A.(2017), Empresas de desarrollo de software en Cochabamba, Impreso en Talleres Gráficos “Kipus”.

Zeballos Gallardo, Gonzalo(2021), La industria del software y su exportación en Bolivia.[En línea]. Consultado: [26 diciembre 2021], Disponible en: [https://ideasparaelfuturo.caf.com/media/3028/bolivia-ganador-4820-9487-la-industria-del-software-y-su-exportacio-n-en-bolivia.pdf]

CHAKRAY(2024). Microservicios. [En Línea]. Recuperado de: [https://www.chakray.com/es/experiencia/microservicios/]

Sanchez, Daniel (2016). Arquitectura de microservicios – Parte 1 Introducción. [En Línea]. Recuperado de: [https://www.enmilocalfunciona.io/arquitectura-microservicios-1/]

NucleusIT (2021). Reflections on the future of microservices. [En línea]. Recuperado de: [https://www.linkedin.com/pulse/reflections-future-microservices-nucleus-it/]

Bibliografía

Shvets Alexander(2021). Sumérgete en los patrones de diseño. Refactoring.Guru.

Blancarte Iturralde, OscarJavier(2020), Introducción a la arquitectura de software: un enfoque práctico. Oscar Blancarte Blog Software Architect.

Effy Oz(). Administración de los sistemas de Información. 5ta. Edición. Cengage Learning Editores.

Macero, Moises (2017). Learn Microservices with spring boot. Editorial Apress.

Richardson, Chris(2019). Microservices Patterns with examples in java. Manning Publications Co.

Tahchiev, Petar. Leme, Felipe. Massol, Vincent. Gregory, Gary. JUnit in action (2da. Edición). Editorial Manning.