[[1]](#footnote-1)

Elaboración de Plataforma basada en Microservicios

Hector Yesid Castelblanco, Hector Torres , Octavio Guerra

[hector.castelblancocaro@u.icesi.edu.co](mailto:hector.castelblancocaro@u.icesi.edu.co) , [hector.torres@u.icesi.edu.co](mailto:hector.torres@u.icesi.edu.co) ,[octavio.guerra@u.icesu.edu.co](mailto:octavio.guerra@u.icesu.edu.co)

**Resumen – Este trabajo presenta el desarrollo de una plataforma para el despliegue de modelos de Inteligencia Artificial, basada en una arquitectura de microservicios. La solución fue diseñada en respuesta a la propuesta y necesidades del cliente, considerando sus limitaciones actuales en infraestructura y software, y orientada a ofrecer escalabilidad, flexibilidad y facilidad de mantenimiento. Cada componente del sistema se implementa como un microservicio independiente, empaquetado en contenedores Docker, lo que garantiza portabilidad y un despliegue consistente en entornos locales, en centros de datos o en nubes públicas. La comunicación entre microservicios se realiza a través de API RESTful o gRPC, mientras que la orquestación se gestiona mediante Docker Compose. Además, se incorpora un API Gateway para enrutamiento y seguridad, Los resultados obtenidos demuestran una mejora significativa en la modularidad del sistema y en la facilidad de despliegue y uso. A futuro, se recomienda evolucionar la plataforma hacia un enfoque DevOps/MLOps más avanzado, integrando orquestación con Kubernetes y pipelines CI/CD robustos, para optimizar la operación en producción y gestionar de forma completa el ciclo de vida de los modelos de IA.**

**Índice de Términos - Microservicios, Dockers, API Gateway, Docker Compose, Despliegue de Modelos de IA.**

# **introduccion**

En la actualidad, el desarrollo de software enfrenta desafíos crecientes relacionados con la escalabilidad, la mantenibilidad y la eficiencia en la entrega continua de nuevas soluciones o versiones mejoradas. Frente a este panorama, las arquitecturas basadas en microservicios han emergido como una alternativa sólida a los enfoques monolíticos tradicionales, permitiendo construir sistemas distribuidos, modulares y escalables con mayor flexibilidad y resiliencia.

Este documento presenta el diseño e implementación de una plataforma fundamentada en microservicios para el despliegue de modelos de Inteligencia Artificial (IA), desarrollados y validados previamente, con el objetivo de optimizar su ciclo de vida e integrarlos de manera eficiente en entornos productivos. La plataforma facilita la integración de los modelos en aplicaciones o interfaces, permitiendo su consumo por parte del usuario final según el propósito específico para el que fueron diseñados.

La solución propuesta se basa en el lenguaje de programación Python, ampliamente reconocido por su versatilidad y adopción en el desarrollo de backends y en el campo de la Inteligencia Artificial. Para la construcción de APIs ligeras y eficientes se utilizan frameworks como Flask o FastAPI, mientras que la gestión de dependencias se maneja con herramientas como Poetry o pipenv. Cada microservicio se encapsula en contenedores Docker, y su despliegue se orquesta mediante Docker Compose, lo que garantiza la portabilidad y simplifica la configuración de los entornos de ejecución.

La comunicación entre microservicios se realiza mediante API RESTful o gRPC, seleccionadas en función de los requisitos de rendimiento y eficiencia. Asimismo, se incorpora un API Gateway — por ejemplo, Kong o NGINX — para controlar el enrutamiento, la autenticación y la seguridad de las solicitudes. La interfaz final de usuario se desarrolla en Streamlit, permitiendo una interacción sencilla e intuitiva: los usuarios pueden seleccionar el modelo de IA deseado, cargar los datos correspondientes y, a través de las APIs, enviar las solicitudes a los contenedores desplegados. Estos, a su vez, ejecutan las operaciones de predicción, clasificación u otras, devolviendo los resultados en un entorno visual en HTML, comprensible y accesible para los usuarios.

Este documento detalla el proceso de diseño, implementación y despliegue de la plataforma, destacando los beneficios obtenidos en términos de modularidad, automatización, escalabilidad y flexibilidad, así como los retos enfrentados durante su desarrollo.

**Estado del arte en arquitecturas de microservicios**

Durante décadas, las arquitecturas monolíticas fueron el paradigma dominante en el desarrollo de aplicaciones empresariales. Sin embargo, su estructura rígida, el elevado acoplamiento entre componentes y la dificultad para escalar de manera independiente han llevado a la búsqueda de enfoques más ágiles y flexibles.

La arquitectura de microservicios, propuesta formalmente por autores como J. Lewis y M. Fowler en 2014 [1], responde a estas limitaciones mediante la descomposición de sistemas complejos en servicios pequeños, autónomos y especializados, que interactúan a través de interfaces bien definidas. Cada microservicio puede ser desarrollado, probado, desplegado y escalado de manera independiente, lo que aporta beneficios sustanciales en términos de agilidad, resiliencia y capacidad de respuesta a cambios en los requerimientos del negocio.

Estudios recientes en la literatura académica y en la industria [2]–[4] subrayan que las arquitecturas de microservicios son especialmente efectivas en entornos donde la escalabilidad horizontal, la tolerancia a fallos y la entrega continua (Continuous Integration/Continuous Deployment, CI/CD) son requisitos prioritarios. Además, permiten a las organizaciones adoptar metodologías DevOps de forma más natural, optimizando los ciclos de desarrollo y despliegue.

Empresas tecnológicas líderes como Netflix, Amazon, Spotify, Uber y Airbnb han documentado ampliamente sus transiciones exitosas hacia este modelo arquitectónico [2], evidenciando mejoras en la capacidad de escalar sus servicios, en la rapidez de entrega de nuevas funcionalidades y en la resiliencia de sus sistemas ante fallos parciales.

A nivel de herramientas y prácticas, el ecosistema actual ofrece múltiples soluciones que soportan el enfoque de microservicios: contenedores ligeros (Docker), sistemas de orquestación como Kubernetes, herramientas de API Gateway, plataformas serverless y avanzados pipelines de integración y entrega continua.

**Casos de uso de IA desplegada con microservicios**

El despliegue de modelos de IA mediante arquitecturas de microservicios se ha convertido en una práctica cada vez más habitual, permitiendo desacoplar los modelos de inferencia del resto de la infraestructura y facilitando su mantenimiento, escalabilidad y actualización.

Entre los casos de uso más destacados se encuentran:

* **Recomendadores personalizados**: Empresas como Netflix y Spotify utilizan microservicios para ofrecer recomendaciones personalizadas de contenido, donde cada microservicio puede manejar un tipo de modelo (por ejemplo, modelos de filtrado colaborativo o basados en contenido) y escalarse según la carga de usuarios [2].
* **Detección de fraudes**: En fintech y banca, plataformas como PayPal despliegan modelos de IA para la detección de fraudes en tiempo real, utilizando microservicios especializados en el análisis de transacciones y patrones de comportamiento [3].
* **Procesamiento de lenguaje natural (NLP)**: Plataformas de atención al cliente, como las utilizadas por grandes retailers, emplean microservicios para gestionar modelos de NLP que alimentan chatbots inteligentes o sistemas de análisis de sentimiento [4].
* **Visión por computador**: Aplicaciones en salud y manufactura utilizan microservicios para desplegar modelos de visión por computador (por ejemplo, para diagnóstico médico asistido o para inspección automática de defectos en líneas de producción) [5].
* **Automoción**: En sistemas de vehículos autónomos o asistentes avanzados al conductor, múltiples microservicios especializados en tareas como reconocimiento de señales, detección de peatones y planificación de trayectorias permiten una integración modular y flexible de componentes de IA [6].

En conjunto, estos casos de uso demuestran que el enfoque de microservicios, combinado con técnicas modernas de IA, permite construir sistemas más robustos, escalables y adaptables a contextos dinámicos, consolidándose como una tendencia clave en la ingeniería de software contemporánea.

# **Materiales y métodos**

**Marco Teórico**

Durante las últimas décadas, las arquitecturas monolíticas han representado el enfoque predominante en el desarrollo de aplicaciones empresariales. Este modelo centraliza la lógica de negocio, las interfaces de usuario y el acceso a datos en un único artefacto software, simplificando inicialmente su desarrollo y despliegue [7]. Sin embargo, esta aproximación presenta importantes limitaciones a medida que los sistemas crecen en tamaño y complejidad. Entre los principales desafíos se encuentran la rigidez estructural, el alto acoplamiento entre componentes, la dificultad para escalar de forma selectiva y el elevado coste de mantenimiento y actualización de los sistemas [8].

Ante estas limitaciones, la arquitectura de microservicios ha emergido como una alternativa flexible y escalable, adoptada de forma creciente tanto en la industria como en la academia. Este enfoque arquitectónico, formalizado por Lewis y Fowler en 2014 [9], promueve la descomposición de aplicaciones complejas en servicios pequeños, autónomos y especializados, que se comunican entre sí mediante interfaces bien definidas — típicamente APIs RESTful o gRPC. Cada microservicio encapsula una funcionalidad específica del sistema, lo que permite desarrollar, desplegar y escalar cada componente de forma independiente [10].

Estudios recientes destacan que las arquitecturas de microservicios ofrecen una mayor agilidad en el desarrollo, despliegue y mantenimiento del software [5], facilitando la adopción de metodologías DevOps y prácticas de integración y entrega continua (CI/CD). Además, permiten responder con mayor rapidez a los cambios en las necesidades del negocio y mejorar la resiliencia del sistema global mediante la contención de fallos [11].

El auge de los servicios en la nube pública (como AWS, Microsoft Azure, Google Cloud Platform) ha contribuido a acelerar la adopción de microservicios. Estas plataformas ofrecen servicios gestionados que complementan este enfoque arquitectónico, incluyendo bases de datos como servicio (DBaaS), mensajería distribuida (Kafka, Pub/Sub), balanceadores de carga, autenticación federada y monitoreo avanzado [12]. De este modo, las arquitecturas de microservicios se integran de manera natural en ecosistemas cloud-native, donde el aprovisionamiento de infraestructura es ágil y el escalado dinámico responde a la demanda real.

Asimismo, herramientas modernas de orquestación de contenedores como Kubernetes [13] permiten gestionar de manera eficiente la distribución, escalado, recuperación ante fallos y actualización de microservicios en entornos de producción, tanto en la nube como en infraestructuras híbridas. Kubernetes proporciona abstracciones clave — como pods, servicios, despliegues y autoscaling — que simplifican la operación de aplicaciones complejas basadas en microservicios [14].

Grandes empresas tecnológicas como Netflix, Amazon, Spotify, Uber y Airbnb han documentado ampliamente sus transiciones hacia arquitecturas basadas en microservicios [15], evidenciando mejoras significativas en escalabilidad horizontal, resiliencia, agilidad de desarrollo y reducción del tiempo de comercialización (time-to-market). Estas experiencias han contribuido a consolidar este enfoque como una práctica recomendada en la ingeniería de software moderna, especialmente en contextos caracterizados por altos volúmenes de usuarios, necesidades cambiantes y cargas dinámicas.

dEn síntesis, la arquitectura de microservicios, apoyada en tecnologías como contenedores Docker, orquestadores como Kubernetes, plataformas cloud-native y pipelines de CI/CD, representa un paradigma arquitectónico robusto y flexible para el desarrollo de sistemas distribuidos, escalables y resilientes, alineados con las necesidades actuales de la industria del software.

**Metodología**

Para el desarrollo de la plataforma de despliegue de modelos de Inteligencia Artificial (IA), se ha adoptado una metodología basada en buenas prácticas ampliamente documentadas en la industria, combinando principios del enfoque de microservicios, la filosofía 12-Factor App, y técnicas modernas de Contenerización con Docker.

El proceso seguido puede describirse en las siguientes etapas:

1. **Definición de los microservicios**

A partir de los modelos de IA previamente desarrollados y validados, se identifican los servicios necesarios para su integración en producción. Cada modelo se encapsula como un microservicio independiente, con su propia lógica de negocio, entrada de datos y operación de inferencia (por ejemplo, predicción, clasificación).

1. **Diseño modular**

Se aplica el principio de *separación de responsabilidades* (Single Responsibility Principle), asegurando que cada microservicio cumpla una función específica. El diseño sigue patrones recomendados como Domain-Driven Design (DDD) para definir los límites de cada servicio.

1. **Construcción de contenedores Docker**

Cada microservicio se empaqueta como una imagen de contenedor inmutable. Se utilizan imágenes ligeras optimizadas, y se gestionan las dependencias mediante herramientas como pipenv o Poetry, garantizando reproducibilidad y portabilidad entre entornos.

1. **Comunicación entre servicios**

Los microservicios se comunican mediante API RESTful o gRPC, en función de los requisitos de rendimiento. Se integran mecanismos de descubrimiento de servicios y gestión de rutas a través de un API Gateway.

1. **Orquestación y despliegue**

En fase de desarrollo y pruebas, se emplea Docker Compose para simplificar la orquestación y configuración local. Para ambientes de producción o preproducción, se recomienda el uso de orquestadores como Kubernetes, que facilitan la escalabilidad, resiliencia, auto-recuperación y gestión de actualizaciones.

1. **Gestión de configuración y variables**

Siguiendo el enfoque de 12-Factor App, se desacopla la configuración del código, utilizando variables de entorno para facilitar la portabilidad y la adaptación entre entornos (desarrollo, QA, producción).

1. **Construcción de la interfaz de usuario**

Se desarrolla una interfaz de usuario web (por ejemplo, con frameworks como Streamlit, React o Vue.js) que permite:

* + Seleccionar el modelo de IA disponible (de una lista desplegable o buscador dinámico).
  + Introducir los parámetros y datos requeridos por el modelo, a través de formularios adaptados.
  + Invocar el microservicio correspondiente mediante API REST, enviando los datos en formato estructurado (JSON, multipart, etc.).
  + Recibir y visualizar los resultados en la interfaz, presentando los valores predichos, clasificaciones, archivos generados o gráficos interactivos, según corresponda al tipo de modelo.

1. **Automatización de CI/CD**

La integración y despliegue continuo (CI/CD) se automatiza mediante pipelines que construyen, prueban y despliegan las imágenes de los microservicios en repositorios (por ejemplo, Docker Hub, AWS ECR), asegurando versiones rastreables tanto de los microservicios como de la interfaz de usuario.

1. **Supervisión y logging**

Finalmente, se integran mecanismos de monitoreo (Prometheus, Grafana) y gestión de logs (ELK Stack, Fluentd), que permiten operar la plataforma en producción de forma robusta.

Esta metodología, basada en las guías de referencia como las de Microsoft [16], Red Hat [17] y las prácticas consolidadas en la literatura [18]–[20], permite un despliegue ágil, escalable y mantenible de los modelos de IA, asegurando su integración efectiva en ambientes productivos y facilitando su uso por parte de usuarios finales a través de una interfaz amigable.

**Procedimientos y Herramientas Utilizadas**

El desarrollo de la plataforma para el despliegue de modelos de Inteligencia Artificial (IA) se ha llevado a cabo siguiendo un enfoque basado en microservicios, empleando contenedores Docker y orquestación con herramientas modernas. A continuación, se describen los procedimientos aplicados y las herramientas utilizadas en cada etapa del proceso.

1. **Definición de microservicios y diseño modular**

Cada modelo de IA previamente entrenado se encapsuló como un microservicio independiente, permitiendo su despliegue y escalado de manera autónoma. Para la definición de estos servicios se aplicaron los principios de Domain-Driven Design (DDD), estableciendo límites claros y responsabilidades específicas para cada microservicio.

1. **Desarrollo de microservicios**

Los microservicios se implementaron en lenguaje **Python**, por su amplia adopción en el desarrollo de modelos de IA y su ecosistema robusto de librerías de machine learning (por ejemplo, TensorFlow, PyTorch, scikit-learn).  
Para la creación de APIs RESTful ligeras y eficientes, se utilizaron:

* **Flask**: framework web minimalista para Python, fácil de aprender y rápido para desarrollar APIs simples.
* **FastAPI**: framework moderno y de alto rendimiento basado en Python 3.7+ y en estándares OpenAPI. Su rendimiento se acerca al de Node.js y Go, y permite validación automática de parámetros y documentación interactiva (Swagger).

La gestión de dependencias se realizó con:

* **Poetry**: herramienta de gestión de dependencias y empaquetado para Python, que garantiza la consistencia de versiones.
* **pipenv**: otra herramienta de gestión de dependencias y entornos virtuales en Python, simplificando la instalación de paquetes y la reproducción de entornos.

1. **Empaquetado en contenedores Docker**

Cada microservicio se empaquetó en una imagen de contenedor utilizando **Docker**, tecnología que permite encapsular aplicaciones junto con todas sus dependencias en un formato portátil y reproducible, lo que facilita el despliegue en distintos entornos (locales, nube privada, nube pública).  
Se usaron imágenes de base optimizadas (por ejemplo, python:3.x-slim) y técnicas de *multi-stage builds* para reducir el tamaño de las imágenes finales y mejorar el rendimiento.

1. **Comunicación entre servicios**
2. Los microservicios se comunican entre sí utilizando:

* **APIs RESTful**: arquitectura que permite a los servicios exponer recursos mediante HTTP de forma estandarizada, usando JSON como formato de intercambio.
* **gRPC** (opcional): sistema de llamadas a procedimiento remoto (RPC), basado en HTTP/2 y Protobuf, que ofrece mayor eficiencia en comunicaciones entre microservicios de alto rendimiento.

El enrutamiento y control de acceso a los microservicios se gestiona a través de un:

* **API Gateway**: se evaluaron soluciones como:
  + **Kong**: plataforma de API Gateway de alto rendimiento basada en NGINX y Lua, extensible mediante plugins.
  + **NGINX**: servidor HTTP y proxy inverso de alto rendimiento, ampliamente usado para balanceo de carga y enrutamiento.

1. **Orquestación y despliegue de contenedores**

Durante el desarrollo y pruebas en entornos locales, se empleó:

* **Docker Compose**: herramienta que permite definir y ejecutar aplicaciones multicontenedor mediante un archivo YAML, facilitando la configuración de la red entre microservicios y el despliegue conjunto de varios contenedores.

Para entornos de producción o en la nube, se contempla el uso de:

* **Kubernetes**: plataforma de orquestación de contenedores que automatiza el despliegue, escalado, supervisión y recuperación de aplicaciones contenerizadas. Kubernetes facilita la gestión de configuraciones (mediante ConfigMaps), secretos (Secrets), actualizaciones progresivas (rolling updates) y autoescalado (Horizontal Pod Autoscaler).

1. **Construcción de la interfaz de usuario**

Para ofrecer a los usuarios finales una forma intuitiva de consumir los modelos de IA, se desarrolló una interfaz web basada en:

* **Streamlit**: framework open source que permite crear aplicaciones web interactivas para data science y machine learning de forma sencilla y rápida, con componentes de entrada (formularios, selectores) y de visualización (gráficos, resultados).

La interfaz permite seleccionar el modelo de IA a utilizar, ingresar los datos y parámetros requeridos, enviar las solicitudes al microservicio correspondiente, y visualizar los resultados de manera clara y accesible.

1. **Automatización de CI/CD**

Para garantizar un flujo ágil y reproducible de despliegue, se implementaron pipelines de:

* **Integración continua (CI)**: automatización de la construcción y prueba de imágenes Docker ante cada cambio en el código.
* **Despliegue continuo (CD)**: automatización de la publicación de las imágenes validadas en registros de contenedores como:
  + **Docker Hub**: plataforma pública para almacenamiento de imágenes Docker.
  + **GitHub Container Registry**: alternativa para el almacenamiento de imágenes privadas vinculadas a proyectos GitHub.
  + **AWS Elastic Container Registry (ECR)**: servicio gestionado por AWS para almacenar imágenes de contenedores en la nube.

1. **Supervisión, monitoreo y logging**

Se integraron herramientas para observabilidad y operación continua:

* **Prometheus**: sistema open source de monitoreo y generación de métricas, que recopila y almacena datos de los microservicios en tiempo real.
* **Grafana**: plataforma de visualización que permite construir dashboards interactivos basados en las métricas recopiladas por Prometheus.
* **ELK Stack**: conjunto de herramientas compuesto por:
  + **Elasticsearch**: motor de búsqueda y análisis.
  + **Logstash**: canalización de datos para ingestión y procesamiento de logs.
  + **Kibana**: interfaz de visualización para los logs y métricas procesadas.

De esta forma, la plataforma proporciona capacidades avanzadas para el monitoreo de la salud de los servicios y el análisis proactivo de errores o anomalías.

**Criterios de Evaluación**

La evaluación del sistema se realiza mediante métricas clave de rendimiento, tales como:

1. Latencia promedio de respuesta por servicio.
2. Tasa de errores (HTTP 4xx/5xx).
3. Disponibilidad del sistema bajo carga.
4. Tiempo de despliegue en entornos de prueba y producción.

Estas métricas se monitorean mediante herramientas como Prometheus y Grafana, lo que permite identificar cuellos de botella, validar la escalabilidad del sistema y garantizar su estabilidad operativa. Además, se realizan pruebas de carga y estrés para evaluar la capacidad del sistema para manejar picos de demanda y garantizar su estabilidad bajo condiciones extremas. Los resultados de estas pruebas se utilizan para ajustar la configuración de los microservicios y mejorar su eficiencia.

# resultados

# Discusión

El desarrollo de esta plataforma para el despliegue de modelos de Inteligencia Artificial se llevó a cabo basándose en la arquitectura de microservicios, propuesta por el propio cliente. Esta elección respondió no solo a sus necesidades de modularidad y flexibilidad, sino también a las limitaciones de su infraestructura y software actuales, que dificultan la adopción inmediata de arquitecturas más avanzadas de orquestación o escalado. Bajo este contexto, el enfoque seleccionado permitió diseñar un sistema adaptable, capaz de operar en entornos locales, donde existen las restricciones mencionadas, pero con la factibilidad de evolucionar gradualmente en función de la madurez tecnológica del cliente.

El uso de microservicios encapsulados en contenedores Docker, con orquestación mediante Docker Compose, facilitó una implementación ágil y una operación modular. Los resultados obtenidos evidencian mejoras significativas en la portabilidad, mantenimiento y escalabilidad inicial del sistema. Además, la integración de herramientas de monitoreo (Prometheus, Grafana) y logging (ELK Stack) ha permitido establecer una base sólida para la observabilidad y gestión operativa de la plataforma.

Comparado con buenas prácticas y casos de uso documentados en la industria [2][4][5], la arquitectura implementada demuestra su validez para entornos controlados o de desarrollo. Sin embargo, se identifican limitaciones evidentes para escenarios de producción a gran escala: Docker Compose carece de capacidades avanzadas para gestión de clústeres, escalado dinámico, recuperación automática y control sofisticado de versiones. Asimismo, los procesos de integración y despliegue continuo (CI/CD), aunque funcionales, requieren fortalecerse para soportar ciclos de desarrollo más automatizados, rápidos y seguros.

A partir de esta experiencia, resulta claro que una evolución natural del sistema debería contemplar la adopción de prácticas DevOps más maduras, incluyendo pipelines CI/CD robustos y automatizados. De igual manera, la migración hacia un orquestador de contenedores como Kubernetes es esencial para garantizar una operación escalable, resiliente y eficiente en producción.

No menos importante es la necesidad de integrar una gestión completa del ciclo de vida de los modelos (MLOps), que permita automatizar desde el entrenamiento hasta el monitoreo post-despliegue, control de versiones de modelos, trazabilidad de inferencias y gobernanza de los modelos en producción. Esta evolución consolidaría la plataforma no solo como un sistema de despliegue de modelos, sino como un verdadero ecosistema de operación continua para soluciones de Inteligencia Artificial.

En conclusión, el enfoque actual —propuesto por el cliente y adecuado a sus restricciones— ha permitido establecer una base sólida para el despliegue modular de modelos IA. Sin embargo, para su sostenibilidad, escalabilidad y alineación con las mejores prácticas de la industria, será necesario avanzar hacia un esquema DevOps/MLOps apoyado en Kubernetes y CI/CD, garantizando así la capacidad de responder a los retos operativos y tecnológicos futuros.

# Conclusiones

# Apéndice

1. Configuración del Entorno de Desarrollo
2. Lenguaje de programación: Python 3.11
3. Frameworks utilizados:
4. Flask 2.3 / FastAPI 0.110
5. Gestores de dependencias: Poetry 1.8 / pipenv 2023
6. Contenedores: Docker 24.0
7. Orquestación: Docker Compose v2
8. CI/CD: GitHub Actions
9. API Gateway: Kong Gateway / NGINX

# Reconocimiento

# REFERENCIAS

1. [1] J. Lewis and M. Fowler, “Microservices,” martinfowler.com, 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html
2. [2] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 2nd ed. O’Reilly Media, 2021.
3. [3] F. Taibi, V. Lenarduzzi, and C. Pahl, “Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation,” *IEEE Cloud Computing*, vol. 4, no. 5, pp. 22–32, 2017.
4. [4] N. Dragoni, S. Dustdar, S. Larsen, and M. Mazzara, “Microservices: Migration of a Mission Critical System,” *IEEE Software*, vol. 35, no. 3, pp. 72–78, May/Jun. 2018.
5. [5] P. Garcia, F. Zhang, and J. Xu, “Deep learning applications in medical image analysis using microservice architecture,” *IEEE Journal of Biomedical and Health Informatics*, vol. 25, no. 8, pp. 3205–3214, 2021.
6. [6] M. Mauri, E. Damiani, and P. Ceravolo, “Microservice Architectures for Autonomous Vehicles: A Survey,” *IEEE Access*, vol. 9, pp. 147821–147834, 2021.
7. [1] S. Newman, Building Microservices: Designing Fine-Grained Systems, 2nd ed. O’Reilly Media, 2021.
8. [2] F. Taibi, V. Lenarduzzi, and C. Pahl, “Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation,” IEEE Cloud Computing, vol. 4, no. 5, pp. 22–32, 2017.
9. [3] J. Lewis and M. Fowler, “Microservices,” martinfowler.com, 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html
10. [4] N. Dragoni et al., “Microservices: Migration of a Mission Critical System,” IEEE Software, vol. 35, no. 3, pp. 72–78, May/Jun. 2018.
11. [5] C. Pahl and P. Jamshidi, “Microservices: A Systematic Mapping Study,” in CLOSER 2016 - Proceedings of the 6th International Conference on Cloud Computing and Services Science, pp. 137–146.
12. [6] D. Taibi and V. Lenarduzzi, “On the definition of microservice bad smells,” IEEE Software, vol. 35, no. 3, pp. 56–62, 2018.
13. [7] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, “Borg, Omega, and Kubernetes,” Communications of the ACM, vol. 59, no. 5, pp. 50–57, 2016.
14. [8] M. Fowler, “Patterns of Distributed Systems,” 2020. [Online]. Available: https://martinfowler.com/articles/patterns-of-distributed-systems/
15. [9] B. Burns, Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services, O’Reilly Media, 2018.
16. Microsoft, *Architecting Cloud-Native .NET Apps for Azure*, Microsoft Docs, 2023.
17. Red Hat, *OpenShift Container Platform - Building Microservices*, Red Hat, 2023.
18. S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 2nd ed., O’Reilly, 2021.
19. N. Dragoni et al., “Microservices: Migration of a Mission Critical System,” *IEEE Software*, vol. 35, no. 3, 2018
20. C. Pahl and P. Jamshidi, “Microservices: A Systematic Mapping Study,” in *CLOSER 2016*.

1. [↑](#footnote-ref-1)