ERICK DANIEL PEÑA CEDEÑO



**Habilidades:**Técnico en multimedia,Programador de software ,trabajo en equipo y comunicación.

**Intereses:** Me apasiona el mundo de la multimedia y la programación de web. Disfruto de combinar mis habilidades técnicas con la creatividad que ofrecen estos campos, y siempre estoy buscando aprender más sobre cómo crear experiencias interactivas y visualmente atractivas.

**Experiencia:**Cuento con un título de tecnólogo en multimedia y tengo experiencia programando en diversos lenguajes, incluyendo C#, Java, HTML, CSS, JavaScript y TypeScript. Actualmente, utilizo Angular para desarrollar aplicaciones web interactivas, lo que me ha permitido aplicar mis conocimientos en proyectos desafiantes y en constante evolución.

**Objetivos:**Mi objetivo a corto plazo es seguir perfeccionando mis habilidades en el desarrollo web, integrando la multimedia y las tecnologías de programación. A largo plazo, me gustaría ser parte de un equipo de desarrollo de software o incluso liderar proyectos creativos que fusionen mis intereses por la programación y la multimedia.

**Características personales**:Soy una persona proactiva y con gran capacidad de adaptación a nuevos retos. Me gusta aprender de manera constante, y disfruto trabajando en equipo. Además, me considero responsable, organizado y con buena capacidad para resolver problemas.

Arquitectura de Software

graficos

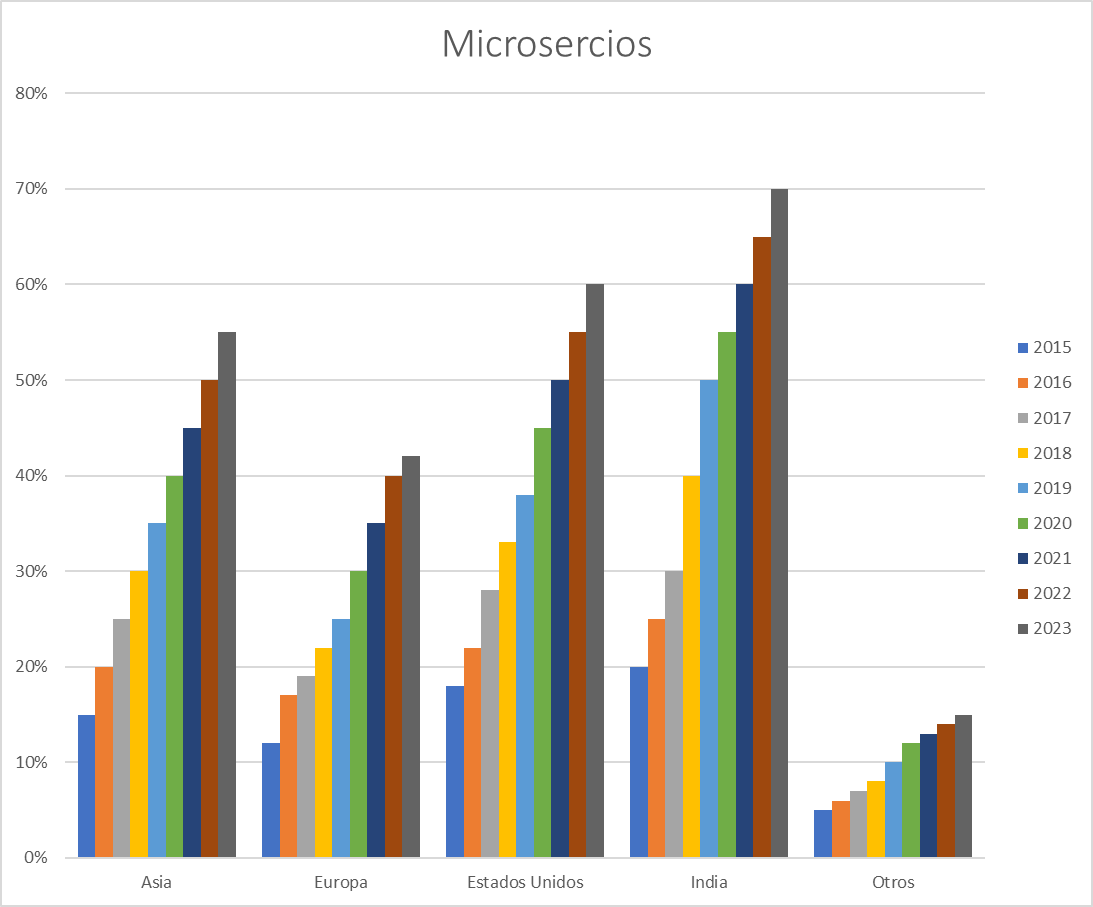
El documento sobre Arquitectura de Software profundiza en la importancia de las Arquitecturas de Referencia de Software (ARS) en el desarrollo de sistemas complejos. Las ARS establecen un marco común que facilita la colaboración entre desarrolladores al definir un conjunto de patrones, principios y lineamientos que guían la construcción del software. Estos patrones no solo mejoran la coherencia y eficiencia del desarrollo, sino que también aseguran que el software sea escalable y sostenible a largo plazo.

Uno de los puntos clave es la utilización de lenguajes de descripción arquitectónica que ayudan a los ingenieros a documentar y comunicar el diseño del sistema. Entre los lenguajes mencionados están Wright, Darwin y C2, cada uno con características que les permiten representar diferentes aspectos de la arquitectura. Estos lenguajes son esenciales para documentar cómo interactúan los componentes y cómo se manejan las dependencias y conexiones entre ellos.

El texto también aborda la relación entre la arquitectura empresarial y los sistemas distribuidos, señalando la necesidad de marcos de referencia sólidos para garantizar que estos sistemas complejos puedan integrarse de manera eficiente y soporten las demandas de escalabilidad y rendimiento.

El documento concluye destacando que los conceptos fundamentales para definir una ARS son el componente (las unidades modulares que realizan funciones específicas dentro del sistema) y el conector (los mecanismos que permiten la interacción entre los componentes). Finalmente, se menciona que la vista arquitectónica es el elemento clave para representar la ARS, ya que proporciona una perspectiva clara y estructurada del sistema, permitiendo a los equipos de desarrollo alinear su trabajo con los objetivos arquitectónicos.

Microservicios



El documento sobre Microservicios explora el desarrollo de una arquitectura de software más flexible y moderna que busca superar las limitaciones de la tradicional arquitectura monolítica. La investigación se enfoca en cómo los microservicios ofrecen una solución más eficiente y adaptable para el desarrollo y mantenimiento de aplicaciones, al permitir que los componentes del sistema funcionen de manera autónoma e independiente.

Una arquitectura monolítica agrupa todas las funciones de una aplicación en un solo bloque de código, lo que puede llevar a problemas significativos a medida que la aplicación crece en tamaño y complejidad. Estos problemas incluyen dificultades en el mantenimiento, el escalamiento y el despliegue de nuevas funciones. El texto argumenta que este enfoque tradicional es rígido y tiende a generar cuellos de botella, ya que cualquier cambio o mejora requiere modificar toda la aplicación, lo que puede provocar interrupciones en las operaciones y la productividad de las organizaciones.

Por otro lado, la arquitectura de microservicios divide una aplicación en pequeñas piezas autónomas llamadas "servicios", que pueden desarrollarse, desplegarse y escalarse de manera independiente. Cada servicio es responsable de una funcionalidad específica y comunica con otros servicios a través de interfaces bien definidas, como APIs. Esta modularidad facilita el \*\*desarrollo ágil\*\*, ya que permite a los equipos de trabajo enfocarse en mejorar y actualizar partes del sistema sin afectar el resto de la aplicación. Además, los microservicios pueden usar tecnologías, lenguajes de programación y bases de datos diferentes según las necesidades de cada componente, lo que proporciona una mayor flexibilidad tecnológica.

El documento destaca varios beneficios clave de los microservicios, incluyendo:

**Escalabilidad:** La capacidad de escalar solo los componentes que lo requieren, en lugar de toda la aplicación.

**Despliegue continuo:** Se pueden realizar cambios en un microservicio sin tener que actualizar o reiniciar todo el sistema.

**Resiliencia:** Si un microservicio falla, no necesariamente afecta a otros, lo que aumenta la disponibilidad de la aplicación.

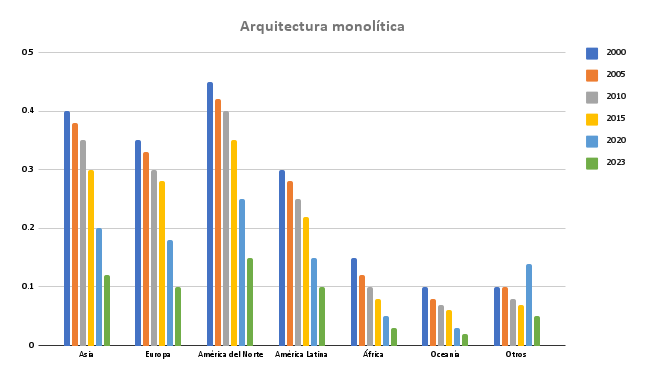
**Desarrollo independiente:** Equipos de desarrollo pueden trabajar en paralelo en diferentes microservicios sin depender de otros equipos.

Sin embargo, el texto también aborda los \*\*desafíos\*\* asociados a esta arquitectura. La adopción de microservicios implica gestionar un \*\*sistema distribuido\*\*, lo cual introduce complejidades en términos de \*\*orquestación\*\*, \*\*seguridad\*\* y \*\*monitorización\*\*. La necesidad de coordinar múltiples servicios autónomos puede derivar en sobrecarga de comunicación entre ellos, lo que puede afectar el rendimiento general si no se maneja correctamente.

Asimismo, se señala la importancia de contar con un equipo de desarrollo altamente capacitado para evitar problemas derivados de la \*\*descentralización\*\* de los servicios, ya que cada equipo podría optar por soluciones técnicas distintas, lo que podría fragmentar el sistema. Además, la implementación de microservicios requiere una infraestructura robusta que soporte la \*\*automatización\*\* del despliegue y la administración de las dependencias entre servicios.

En resumen, la arquitectura de microservicios es presentada como una solución moderna y flexible que permite un desarrollo más ágil y eficiente de aplicaciones. Aunque ofrece importantes ventajas en términos de escalabilidad y mantenimiento, su implementación conlleva una serie de retos relacionados con la complejidad del sistema distribuido y la coordinación de equipos de desarrollo.

**Arquitectura monolítica**

****

**3**

**grafico**

El artículo sobre la **arquitectura monolítica** analiza los beneficios y riesgos asociados a la migración de un sistema monolítico a una arquitectura de microservicios. La arquitectura **monolítica** tradicional consiste en un único código base que abarca todas las funcionalidades de una aplicación. Si bien es más sencilla de gestionar en etapas iniciales de desarrollo, a medida que la aplicación crece, surgen desafíos relacionados con la escalabilidad, mantenimiento y velocidad de desarrollo.

Uno de los principales problemas con una arquitectura monolítica es que cualquier cambio o mejora, incluso en una parte pequeña del sistema, requiere que se despliegue y pruebe todo el sistema completo. Este enfoque puede llevar a tiempos de inactividad y errores operativos, ya que las actualizaciones menores pueden impactar a toda la aplicación. A medida que la aplicación se vuelve más compleja, el monolito tiende a ser menos flexible, y su mantenimiento puede volverse cada vez más costoso y propenso a errores.

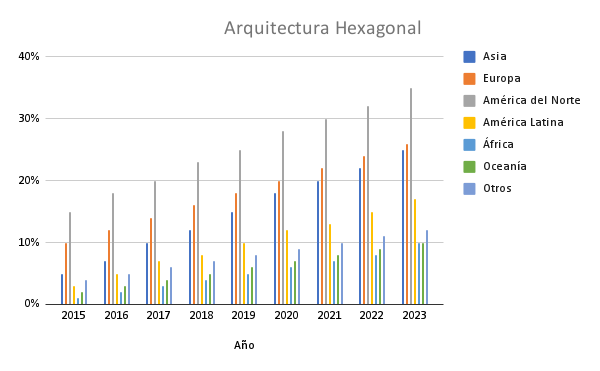
El artículo destaca los **beneficios** de migrar de una arquitectura monolítica a una basada en **microservicios**, como la flexibilidad para actualizar y desplegar partes específicas del sistema de manera independiente, lo que mejora el rendimiento y la capacidad de escalar servicios individuales. Sin embargo, también señala los **riesgos** asociados a la migración, como la complejidad de gestionar múltiples servicios, la necesidad de una sólida infraestructura para automatización, y los posibles **desafíos de seguridad** que surgen al descentralizar las funciones en diferentes servicios.

Migrar de un sistema monolítico a microservicios es un proceso complicado que debe abordarse de forma gradual y meticulosa. El artículo sugiere que un **buen punto de partida** es desarrollar nuevas funcionalidades como microservicios independientes o revisar las partes más recientes del código para evaluar si pueden ser transformadas en microservicios. Este enfoque permite realizar una **migración incremental**, donde se valida constantemente que el sistema mantenga su comportamiento esperado sin introducir errores.

El documento también menciona la importancia de **identificar las "costuras"** dentro del código monolítico. Las costuras son áreas del código que pueden ser separadas y convertidas en microservicios sin alterar el funcionamiento global de la aplicación. El proceso de **migración incremental** asegura que, aunque una parte del sistema ya esté en microservicios, el resto del sistema sigue funcionando bajo el modelo monolítico hasta que todo el sistema pueda ser migrado completamente.

Uno de los puntos clave es que durante este proceso de migración es esencial **detener la adición de nuevo código** al monolito, ya que esto solo complicaría el proceso. En su lugar, se debe trabajar en **aislar y mejorar** las partes del código que puedan convertirse en microservicios, siempre validando que la migración no afecte la funcionalidad existente.

**Arquitectura Hexagonal**



**4**

El artículo sobre HexaGame detalla una plataforma en línea diseñada para coleccionistas de videojuegos, desarrollada utilizando AngularJS en el frontend y SpringBoot en el backend. La aplicación tiene como objetivo facilitar la gestión de colecciones de videojuegos, permitiendo a los usuarios llevar un control detallado de sus juegos, comparar precios y crear listas de deseos. Además, hay planes para expandir la plataforma a dispositivos móviles utilizando SwiftUI para iOS y Kotlin para Android, lo que ampliará su alcance y accesibilidad.

Un aspecto clave del desarrollo de HexaGame es su fundamento en la Arquitectura Hexagonal, también conocida como Arquitectura de Puertos y Adaptadores. Este enfoque arquitectónico es muy valorado por su capacidad para separar las preocupaciones dentro de una aplicación, promoviendo una independencia entre el núcleo de la lógica de negocio y las tecnologías externas, como bases de datos, interfaces de usuario o APIs externas. Este principio de separación permite que el sistema sea más flexible y modular, ya que los componentes pueden desarrollarse y modificarse de forma aislada sin afectar al resto de la aplicación.

La Arquitectura Hexagonal organiza el sistema en torno a tres capas principales:

Capa del Dominio (Core): Contiene la lógica central de la aplicación, es decir, las reglas de negocio y las entidades principales. Esta capa es independiente de cualquier tecnología o infraestructura, lo que significa que no sabe nada sobre cómo los datos se almacenan o cómo se interactúa con el usuario.

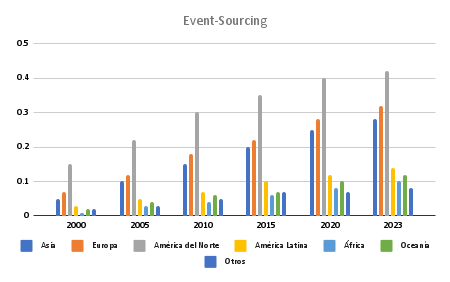
Capa de Puertos: Define las interfaces a través de las cuales la aplicación interactúa con el mundo exterior. Los puertos actúan como contratos que especifican lo que la aplicación necesita de la infraestructura o cómo los servicios externos deben interactuar con el sistema. Por ejemplo, un puerto puede especificar cómo un servicio debe enviar y recibir datos desde una API externa.

Capa de Adaptadores: Es responsable de implementar las interfaces definidas en los puertos, conectando la lógica del dominio con tecnologías externas como bases de datos, interfaces de usuario o servicios externos. Los adaptadores permiten que la infraestructura cambie sin afectar el núcleo de la aplicación, ya que solo necesitan cumplir con los contratos especificados en los puertos.

El proyecto HexaGame utiliza un proyecto Maven multimódulo en su backend, donde cada módulo está diseñado para cumplir con un rol específico dentro del diagrama de la Arquitectura Hexagonal. Esta estructura modular asegura que cada parte del sistema tenga una función clara y que las dependencias entre los módulos estén bien definidas, lo que hace que el proyecto sea más mantenible y escalable a medida que crece.

Esta arquitectura también está muy alineada con los principios del Diseño Dirigido por el Dominio (DDD), que se enfoca en modelar el software en torno al negocio y las necesidades del dominio en lugar de las limitaciones tecnológicas. Al combinar la Arquitectura Hexagonal con DDD, HexaGame logra una estructura de software robusta y flexible que facilita tanto el desarrollo como el mantenimiento a largo plazo.

**Event-Sourcing**

****

**5**

El trabajo analiza el desarrollo de una aplicación que utiliza los patrones de diseño Event-Sourcing y CQRS (Command Query Responsibility Segregation), destacando cómo estos patrones permiten una gestión eficiente de eventos en sistemas complejos. El objetivo principal es implementar una estructura desacoplada que separe claramente el tratamiento, almacenamiento y gestión de los eventos, lo que aporta flexibilidad y escalabilidad a la aplicación.

En el subsistema de escritura, los eventos son capturados por componentes llamados EventSourcingHandlers y almacenados en una base de datos especializada conocida como EventStore. Este almacenamiento basado en eventos permite que cada cambio en el estado del sistema se guarde como un evento separado, lo que facilita la reconstrucción del estado completo del sistema en cualquier momento. Cada vez que ocurre una acción, como la creación o modificación de un anuncio inmobiliario, se genera un evento que se persiste en el EventStore.

En el subsistema de lectura, un componente llamado EventHandler en la clase InmuebleRepositoryProjection captura los eventos y actualiza la versión más reciente del catálogo de inmuebles en una base de datos de lectura. Este diseño garantiza que la información esté siempre actualizada para los usuarios que consultan el catálogo, permitiendo una separación clara entre las operaciones de escritura y lectura, lo que es una de las principales ventajas del patrón CQRS. De esta forma, las consultas se pueden optimizar de manera independiente de las operaciones de escritura, mejorando el rendimiento del sistema.

El proyecto está contextualizado en un portal inmobiliario, donde se gestionan eventos relacionados con la creación, modificación y eliminación de anuncios de propiedades. Este entorno es ideal para aplicar Event-Sourcing, ya que los sistemas inmobiliarios suelen tener un volumen elevado de cambios y actualizaciones en tiempo real, y es crucial mantener un historial detallado de todas las operaciones realizadas.

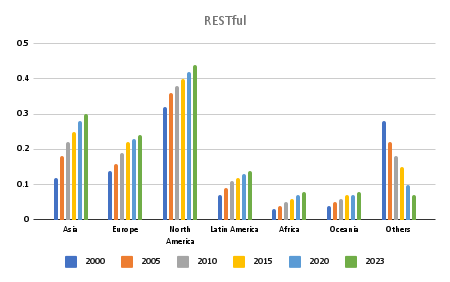
El patrón Event-Sourcing es particularmente útil en escenarios como este porque permite:

Trazabilidad completa: Al almacenar cada cambio de estado como un evento separado, es posible reconstruir el historial completo de cada anuncio de inmueble, lo que permite auditar el sistema de manera efectiva.

Desempeño mejorado en consultas: Al desacoplar las operaciones de escritura y lectura mediante CQRS, las consultas pueden ser mucho más rápidas y eficientes, ya que se puede diseñar la base de datos de lectura de manera optimizada para las necesidades de consulta.

Escalabilidad: La aplicación puede manejar grandes volúmenes de eventos y actualizaciones sin perder rendimiento, ya que las operaciones de escritura no interfieren con las de lectura

**RESTful**

****

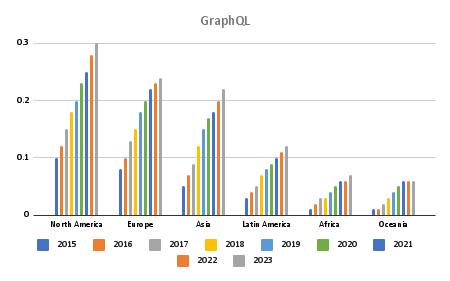
El proyecto 'House Finder' es una innovadora aplicación web diseñada para facilitar la búsqueda de inmuebles en las principales ciudades de Colombia. Con el objetivo de centralizar las ofertas inmobiliarias dispersas en diversas plataformas, 'House Finder' permite a los usuarios acceder a un catálogo extenso de propiedades desde una única interfaz. La aplicación emplea la metodología ágil Kanban, lo que permite un manejo flexible y eficiente de las tareas durante el desarrollo del proyecto. La arquitectura de esta aplicación está dividida entre un backend desarrollado en Java y un frontend implementado en Angular, asegurando una experiencia de usuario fluida y una robusta gestión de datos.

Una de las tecnologías clave detrás de 'House Finder' es la arquitectura RESTful. RESTful (Representational State Transfer) es un estilo arquitectónico ampliamente utilizado para diseñar interfaces de programación de aplicaciones (APIs) que facilitan la interacción entre sistemas a través de la web. Esta arquitectura aprovecha el protocolo HTTP para la comunicación y utiliza formatos de intercambio de datos como JSON y XML. Gracias a RESTful, es posible desarrollar APIs que sean escalables, eficientes y seguras, permitiendo una interacción continua y confiable entre distintas aplicaciones.

En Colombia, RESTful es una tecnología esencial en el desarrollo de sistemas que requieren compartir datos de manera eficiente y segura. 'House Finder' utiliza esta arquitectura para integrar información proveniente de las principales plataformas inmobiliarias del país, como Metro Cuadrado, Finca Raíz y Ciencuadras. Mediante el uso de estas APIs RESTful, la aplicación extrae, procesa y presenta las mejores ofertas de inmuebles de manera automatizada. Esto no solo agiliza el proceso de búsqueda para los usuarios, sino que también garantiza que obtengan la información más actualizada y relevante en tiempo real.

El proyecto 'House Finder' ejemplifica cómo la combinación de tecnologías modernas como Java, Angular y RESTful puede transformar la experiencia del usuario al buscar bienes raíces, proporcionando una plataforma eficiente, intuitiva y centralizada que aborda una necesidad crítica en el mercado colombiano.

**GraphQL**

****

**7**

El documento presentado por Zamia Marlene Guitarra De la Cruz para la obtención del título de Ingeniera en Software en la Universidad Técnica del Norte se enfoca en el desarrollo de una aplicación backend para la gestión de transferencias utilizando tecnologías modernas como microservicios, GraphQL, REST y contenedores Docker, integrando una arquitectura DevOps. Este trabajo refleja la importancia creciente de los enfoques modernos en la creación de aplicaciones que optimicen tanto la flexibilidad como la escalabilidad en entornos de producción.

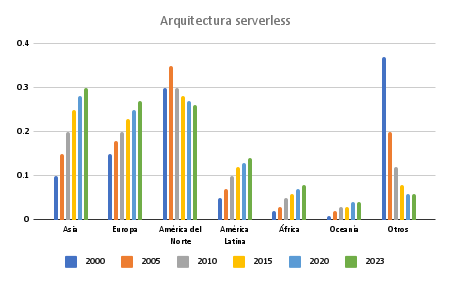
GraphQL, uno de los pilares de este proyecto, es un lenguaje de consulta que ha ganado popularidad como alternativa a la arquitectura REST, gracias a su capacidad de definir de manera precisa los datos que un cliente requiere de un servidor. Desarrollado por Facebook en 2012 e implementado públicamente en 2015, GraphQL permite a los desarrolladores solicitar solo los datos necesarios, evitando el problema común en REST, donde se suele recibir un documento JSON con todos los campos, incluso si solo se necesitan unos pocos.

Una de las grandes ventajas de GraphQL es su flexibilidad para manejar datos dinámicos. A través de un único endpoint, es posible acceder a diferentes recursos sin la necesidad de crear múltiples rutas, lo que simplifica la estructura de la API. Además, GraphQL facilita tres tipos principales de operaciones: **consultas** (queries), **mutaciones** (mutations) y **suscripciones** (subscriptions). Las consultas permiten recuperar datos específicos de la base de datos o de los servicios; las mutaciones son útiles para modificar o crear nuevos datos; y las suscripciones permiten la actualización en tiempo real, ya que el servidor puede enviar datos a los clientes en respuesta a ciertos eventos.

El proyecto también aborda la implementación de microservicios, un enfoque arquitectónico en el que una aplicación se divide en pequeños servicios independientes que se comunican entre sí. Al utilizar contenedores Docker, estos microservicios pueden ser desplegados y escalados de manera más eficiente, fomentando una arquitectura ágil y compatible con las prácticas DevOps, que integran el desarrollo y la operación en un ciclo continuo de mejora.

En resumen, el trabajo de Guitarra De la Cruz no solo explora la aplicación práctica de tecnologías como GraphQL, REST y Docker en el desarrollo de una aplicación backend, sino que también destaca la importancia de las arquitecturas modernas y las metodologías DevOps para lograr sistemas más flexibles, eficientes y fáciles de mantener en el tiempo.

**Arquitectura serverless**

****

**8**

El trabajo de diseño e implementación de un chatbot con arquitectura serverless explora cómo aprovechar tecnologías modernas para crear una aplicación sin la necesidad de administrar servidores tradicionales. Utilizando Amazon Web Services (AWS) y Dialogflow, el proyecto implementa una solución de chatbot que se ejecuta en la nube, destacando la eficiencia y escalabilidad de la arquitectura serverless.

Arquitectura serverless se refiere a un modelo en el que los desarrolladores crean aplicaciones sin gestionar directamente la infraestructura subyacente. En lugar de preocuparse por la configuración, escalabilidad o mantenimiento de servidores, la infraestructura es gestionada automáticamente por el proveedor de servicios en la nube, en este caso, Amazon Web Services (AWS). AWS se encarga de ejecutar el código, asignar recursos de manera dinámica y manejar la escalabilidad en función de la demanda, lo que permite a los desarrolladores concentrarse en escribir código.

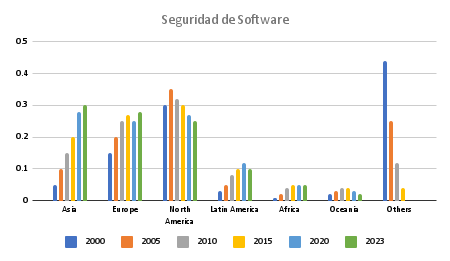
En este proyecto, el backend del chatbot está alojado en AWS, lo que significa que la lógica de la aplicación y las funciones de procesamiento se ejecutan en la nube. AWS Lambda, el servicio serverless de Amazon, ejecuta funciones en respuesta a eventos, como las solicitudes de los usuarios que interactúan con el chatbot. Este enfoque elimina la necesidad de administrar servidores constantemente, reduciendo costos y complejidad operativa.

Para el procesamiento del lenguaje natural, se utiliza Dialogflow, una plataforma desarrollada por Google que facilita la creación de interfaces de conversación. Dialogflow permite entrenar al chatbot para interpretar el lenguaje humano y responder de manera coherente a las consultas de los usuarios. La combinación de AWS y Dialogflow potencia el chatbot al integrar procesamiento eficiente con una infraestructura que escala automáticamente.

En cuanto al frontend, se desarrolló utilizando **Vue.js**, un framework JavaScript ligero que se emplea para crear interfaces de usuario interactivas. El código del frontend se almacena en **Amazon S3**, un servicio de almacenamiento en la nube de AWS que proporciona alta disponibilidad y seguridad. S3 se integra fácilmente con otros servicios de AWS, lo que lo convierte en una opción ideal para alojar aplicaciones web estáticas.

Para la autenticación de usuarios, se utiliza **AWS Cognito**, un servicio que simplifica el proceso de autenticación y autorización en aplicaciones. Cognito permite a los usuarios registrarse, iniciar sesión y acceder al chatbot de manera segura. Además, ofrece soporte para la autenticación a través de redes sociales, lo que facilita el acceso a los usuarios finales.

**Seguridad de Software**

****

**9**

El artículo se centra en el análisis de metodologías de desarrollo de software seguro con enfoque en propiedades ágiles, haciendo especial hincapié en Microsoft Security Development Lifecycle (SDL) y Building Security In Maturity Model (BSIMM). A través de una investigación hermenéutica, descriptiva y heurística, se examina cómo estas metodologías pueden aplicarse en el ciclo de vida del desarrollo de software y se evalúa su capacidad para integrarse en entornos ágiles.

El concepto de **seguridad de software** se refiere a la protección de las aplicaciones y sus datos contra amenazas y vulnerabilidades durante todo su ciclo de vida, desde la concepción y el diseño hasta el mantenimiento y la actualización. Las metodologías ágiles, por su parte, se centran en el desarrollo iterativo y la entrega rápida de software, lo que plantea desafíos para la implementación de prácticas de seguridad que tradicionalmente pueden verse como rígidas o lentas.

### **Metodologías Analizadas:**

1. **Microsoft Security Development Lifecycle (SDL):** Microsoft SDL es un conjunto de prácticas de desarrollo de software diseñado para ayudar a los desarrolladores a construir aplicaciones seguras y a mitigar vulnerabilidades de manera proactiva. Esta metodología introduce la seguridad en cada fase del ciclo de vida del desarrollo, desde la fase de requisitos hasta la de mantenimiento. SDL es compatible con prácticas ágiles, ya que permite la incorporación de medidas de seguridad de manera incremental y ajustada a los ciclos iterativos de desarrollo.
2. **Building Security In Maturity Model (BSIMM):** BSIMM es un marco de referencia que describe las prácticas de seguridad observadas en una amplia variedad de organizaciones. A diferencia de SDL, que es una guía prescriptiva, BSIMM proporciona un enfoque descriptivo, permitiendo a las organizaciones medir su madurez en seguridad de software. Este modelo se basa en la observación de las mejores prácticas de seguridad implementadas en empresas líderes y permite a los equipos de desarrollo adaptar las medidas de seguridad de acuerdo con sus necesidades y el contexto ágil.

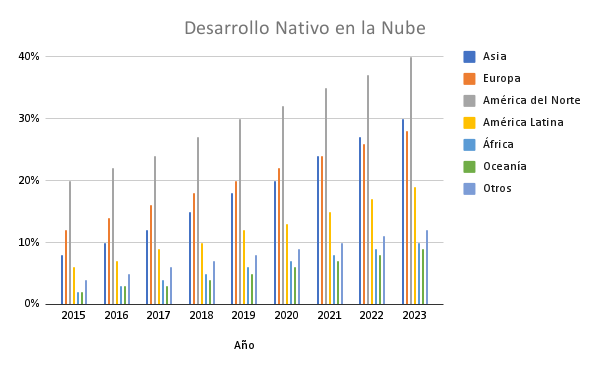
### **Enfoque de Investigación:**

La investigación hermenéutica, descriptiva y heurística realizada en el artículo permitió una comparación detallada entre Microsoft SDL y BSIMM, evaluando su eficacia en la promoción de seguridad dentro del desarrollo ágil. El análisis concluye que ambas metodologías son compatibles con entornos ágiles, ya que permiten la integración de prácticas de seguridad sin interferir con la naturaleza iterativa y rápida del desarrollo ágil.

### **Conclusión:**

El artículo concluye que, aunque existen varias metodologías de desarrollo de software seguro, Microsoft SDL y BSIMM son las únicas que poseen propiedades ágiles, lo que les permite adaptarse efectivamente a los ciclos iterativos y rápidos del desarrollo ágil. Esto es crucial para los equipos que desean equilibrar la velocidad y flexibilidad del desarrollo ágil con la necesidad de proteger el software contra amenazas y vulnerabilidades desde las primeras etapas del proceso de desarrollo.

**Desarrollo nativo en la nube**

****

**10**

El informe presentado por José Alexander Suaza Montes describe su práctica profesional en la empresa Intergrupo Software One, donde se enfocó en la modernización y automatización de procesos empresariales, utilizando tecnologías de **SharePoint** y herramientas de **Cloud Computing**. El documento proporciona una visión detallada de las actividades realizadas durante su experiencia, así como de los aprendizajes obtenidos en torno al desarrollo nativo en la nube.

Desarrollo nativo en la nube se refiere al diseño y construcción de aplicaciones que están optimizadas para ejecutarse en plataformas de nube, aprovechando todas las ventajas que estas ofrecen, como la escalabilidad, la flexibilidad, y la disponibilidad global. Este enfoque implica utilizar servicios y tecnologías en la nube desde la etapa de planificación hasta el despliegue y mantenimiento de las aplicaciones.

### Actividades Realizadas:

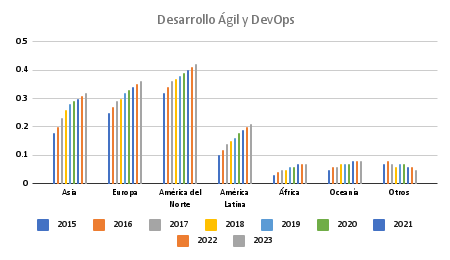
Durante su práctica, Suaza Montes participó en la modernización de procesos empresariales mediante el uso de **SharePoint**, una plataforma de colaboración y gestión de documentos que se integra estrechamente con otras soluciones en la nube de Microsoft. SharePoint, cuando se implementa en un entorno nativo en la nube, permite a las empresas automatizar flujos de trabajo, gestionar contenido de manera eficiente y mejorar la colaboración entre equipos distribuidos geográficamente.

El enfoque en la **automatización de procesos** empresariales permitió a la organización reducir tareas repetitivas y mejorar la eficiencia operativa. Esto se logró mediante la creación de soluciones personalizadas en SharePoint y su integración con otras herramientas de Microsoft en la nube, como **Power Automate**, que facilita la creación de flujos de trabajo automatizados sin necesidad de escribir código.

### Conclusión:

El informe refleja el valor de la experiencia práctica en el ámbito del desarrollo nativo en la nube y la importancia de la modernización de procesos empresariales mediante la utilización de herramientas en la nube, como SharePoint y Azure. La práctica profesional permitió a Suaza Montes comprender cómo la tecnología en la nube puede transformar digitalmente las operaciones empresariales, mejorando la productividad y optimizando recursos.

**Desarrollo Ágil** y **DevOps**

****

**11**

El artículo aborda el Desarrollo Ágil y DevOps, dos enfoques complementarios que han revolucionado la manera en que se gestionan y entregan proyectos de software en la actualidad. Ambos se enfocan en mejorar la eficiencia y la calidad en el desarrollo, pero lo hacen desde diferentes perspectivas.

### Desarrollo Ágil:

El Desarrollo Ágil se refiere a un conjunto de metodologías que priorizan la **colaboración** entre los equipos, la **iteración rápida** y la capacidad de **adaptarse a los cambios** durante el proceso de desarrollo. Entre las metodologías ágiles más conocidas se encuentran **Scrum** y **Kanban**.

* **Scrum**: Es una metodología ágil que divide el trabajo en ciclos cortos llamados **sprints**, generalmente de dos a cuatro semanas. Al final de cada sprint, se entrega una versión funcional del producto, lo que permite recibir retroalimentación constante y ajustar el desarrollo en función de las necesidades del cliente.
* **Kanban**: En lugar de trabajar en ciclos de tiempo definidos, Kanban utiliza un enfoque visual para gestionar el flujo de trabajo a través de tableros. Las tareas se mueven a través de diferentes etapas del proceso, lo que permite una mejora continua y una visión clara del estado del proyecto.

El Desarrollo Ágil es ideal para proyectos donde los **requisitos son cambiantes o no están completamente definidos** desde el inicio, ya que permite un ajuste continuo en función de la evolución de las necesidades del cliente o del mercado. Sin embargo, su implementación requiere una fuerte cultura de colaboración y un equipo dispuesto a adaptarse rápidamente.

### DevOps:

Por otro lado, **DevOps** es una filosofía que busca **integrar el desarrollo (Dev)** y las **operaciones (Ops)** para mejorar la colaboración entre estos dos equipos, que tradicionalmente han trabajado de manera independiente. DevOps se centra en la **automatización** y la **entrega continua** a lo largo de todo el ciclo de vida del software, desde el desarrollo hasta el despliegue y mantenimiento.

Los principales objetivos de DevOps incluyen:

* **Entrega Continua**: Automatización del proceso de integración y entrega de código, permitiendo despliegues rápidos y frecuentes.
* **Monitoreo y Feedback Continuo**: Evaluación constante del rendimiento de las aplicaciones y retroalimentación en tiempo real para detectar y corregir problemas de manera proactiva.
* **Colaboración Fluida**: Fomentar una cultura de colaboración entre desarrolladores y operadores, lo que permite un ciclo de vida de desarrollo más eficiente y menos fricción en las etapas de despliegue.

DevOps es una extensión natural de los principios ágiles, pero su enfoque se amplía para abarcar todo el ciclo de vida del software, incluyendo la operación y el mantenimiento. La automatización es una piedra angular de DevOps, ya que permite implementar cambios en el software de manera más rápida y segura.

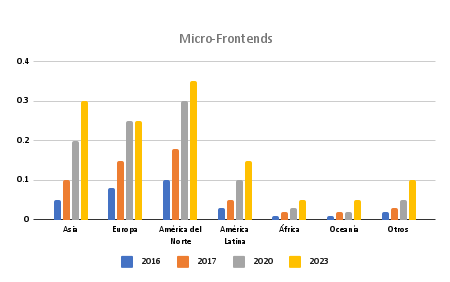
### Comparación y Desafíos:

Si bien ambos enfoques promueven la eficiencia y la mejora continua, también presentan desafíos específicos. En el caso del Desarrollo Ágil, uno de los retos más comunes es mantener la cohesión del equipo y evitar que la falta de planificación a largo plazo cause desorden en el proyecto. Por su parte, la implementación de DevOps requiere una fuerte inversión en herramientas de automatización y un cambio cultural significativo dentro de la organización para romper las barreras tradicionales entre desarrollo y operaciones.

### Conclusión:

El artículo concluye que tanto el Desarrollo Ágil como DevOps tienen un impacto profundo en la forma en que se desarrolla y entrega el software en la actualidad. Ambos enfoques ayudan a las organizaciones a adaptarse mejor a las demandas del mercado, mejorar la calidad del software y acelerar el tiempo de entrega. Sin embargo, para aprovechar al máximo sus beneficios, es crucial entender sus principios, adaptarlos al contexto del proyecto y enfrentar sus desafíos inherentes de manera proactiva.

### **Micro-Frontends**



12

El artículo examina la implementación de **micro-frontends** en el Banco de Bogotá como una estrategia para mejorar la mantenibilidad y escalabilidad de los productos digitales, especialmente en el contexto de una arquitectura basada en microservicios para la parte del frontend. A lo largo del documento, se destaca cómo esta tecnología busca resolver las limitaciones asociadas con arquitecturas monolíticas tradicionales.

### Definición y Características de los Micro-Frontends:

Los **micro-frontends** extienden el concepto de microservicios, pero aplicados a la interfaz de usuario. En lugar de tener una única aplicación monolítica para el frontend, se divide en componentes más pequeños e independientes, donde cada uno se encarga de una parte específica de la interfaz de usuario. Estos componentes pueden ser desarrollados y desplegados de manera autónoma por diferentes equipos, lo que facilita la **modularidad**, la **descentralización** y la **responsabilidad única**.

Cada micro-frontend puede estar asociado a un área funcional específica de la aplicación, como la gestión de cuentas o el procesamiento de pagos, y puede utilizar tecnologías diferentes, siempre que sigan los estándares establecidos para asegurar la cohesión en la experiencia del usuario.

### Problemática de la Arquitectura Monolítica:

El artículo señala que la arquitectura anterior del Banco de Bogotá estaba basada en un único componente monolítico, lo que generaba varios problemas. Entre ellos se destaca la **dificultad para mantener y escalar** el sistema, así como los desafíos para realizar integraciones continuas. En un sistema monolítico, cualquier cambio en una parte de la aplicación requería la modificación y el despliegue del sistema completo, lo que implicaba riesgos mayores y ciclos de desarrollo más largos.

Además, la falta de **responsabilidad única** en el código causaba complicaciones, ya que los equipos de desarrollo debían coordinarse estrechamente para evitar que los cambios en una funcionalidad afectaran otras áreas del sistema.

### Implementación de Micro-Frontends:

La implementación de micro-frontends en el Banco de Bogotá fue propuesta como una solución para enfrentar estos desafíos. Esta arquitectura permite que cada componente del frontend se modifique y despliegue de manera independiente, sin necesidad de afectar a toda la aplicación. Esto es particularmente útil para fomentar **pequeños despliegues continuos** y mejorar la **responsabilidad individual** en cada unidad funcional del software.

El artículo también analiza la **segregación de componentes** en micro-frontends, evaluando cómo esta estrategia puede impactar tanto en la **velocidad de desarrollo** como en la **calidad del software**. Aunque la separación en micro-frontends ofrece ventajas claras en términos de mantenibilidad y escalabilidad, también plantea preguntas sobre la eficiencia y la complejidad adicional que puede introducir al tener múltiples equipos trabajando en partes diferentes de la aplicación.

### Ventajas de los Micro-Frontends:

Entre las principales ventajas de los micro-frontends mencionadas en el artículo se incluyen:

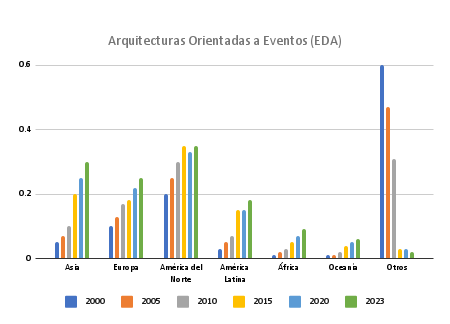
* **Escalabilidad mejorada**: Al poder escalar cada componente de manera independiente, es posible optimizar el rendimiento de áreas específicas de la aplicación sin necesidad de modificar el sistema completo.
* **Mantenibilidad**: La división en componentes pequeños facilita la gestión del código y permite realizar cambios con menos riesgo de introducir errores en otras partes de la aplicación.
* **Despliegue continuo**: Cada micro-frontend puede ser desplegado de forma autónoma, lo que facilita la implementación de actualizaciones y la integración continua.

### Conclusión:

El artículo concluye que la adopción de micro-frontends en el Banco de Bogotá podría resolver muchos de los problemas asociados con la arquitectura monolítica anterior, ofreciendo una solución que mejora la mantenibilidad, la escalabilidad y la velocidad de desarrollo. Sin embargo, también se reconoce que la implementación de micro-frontends no está exenta de desafíos, especialmente en términos de coordinar equipos de desarrollo y asegurar una experiencia de usuario consistente.

En resumen, los micro-frontends presentan una evolución prometedora en la arquitectura de aplicaciones web, permitiendo que grandes organizaciones, como el Banco de Bogotá, enfrenten los desafíos del desarrollo de software moderno de manera más eficiente y escalable.

**Arquitecturas Orientadas a Eventos (EDA**

****

**13**

El artículo ofrece un análisis detallado de las Arquitecturas Orientadas a Eventos (EDA**, por sus siglas en inglés)**, enfocándose en su funcionamiento, ventajas, y su comparación con otros patrones arquitectónicos predominantes. A través de esta evaluación, se destacan las capacidades de las EDA para procesar eventos en tiempo real, lo que resulta esencial para sistemas que requieren respuestas rápidas y eficientes ante cambios en su entorno.

### Definición de Arquitectura Orientada a Eventos (EDA):

Una **Arquitectura Orientada a Eventos** es un patrón de diseño en el que las decisiones del sistema se desencadenan por la generación de eventos. Un **evento** es cualquier cambio significativo en el estado de un sistema, como la llegada de un mensaje, una actualización de datos o una alerta de sistema. Estos eventos son detectados por el sistema, que reacciona a ellos en tiempo real.

En una EDA, los elementos clave incluyen:

* **Productores**: Son las entidades que generan eventos. Pueden ser aplicaciones, sensores o usuarios que interactúan con el sistema.
* **Canal**: Es el medio a través del cual los eventos se transmiten a los consumidores. Puede ser una cola de mensajes, un servicio de eventos o cualquier otra infraestructura de comunicación.
* **Consumidores**: Son las entidades que reciben y procesan los eventos. Pueden ser otros sistemas, bases de datos o aplicaciones que realizan acciones en respuesta a los eventos.
* **Actividad**: Se refiere a las acciones que se toman en función de los eventos procesados. Esto puede incluir actualizaciones de estado, generación de nuevos eventos o respuestas automáticas.

### Comparación con Otros Patrones Arquitectónicos:

El artículo compara las EDA con otros patrones arquitectónicos comunes, como las arquitecturas monolíticas y las basadas en microservicios. A diferencia de las arquitecturas tradicionales, donde las interacciones suelen estar basadas en solicitudes y respuestas sincrónicas, una EDA **procesa los eventos de manera asíncrona**. Esto permite que el sistema responda a los eventos en tiempo real, sin tener que esperar a que se completen otras tareas, lo que mejora la **escalabilidad** y la **eficiencia**.

Una EDA también ofrece una **alta flexibilidad**, ya que permite agregar nuevos consumidores o productores de eventos sin afectar la arquitectura existente. Esto la hace ideal para aplicaciones que requieren una alta modularidad y la capacidad de adaptarse rápidamente a cambios en el entorno.

### Requisitos No Funcionales: Seguridad y Rendimiento:

El artículo destaca dos requisitos no funcionales esenciales para cualquier implementación de EDA: **seguridad** y **rendimiento**.

* **Seguridad**: Dado que las EDA manejan grandes volúmenes de datos en tiempo real, la seguridad es un aspecto crucial. Es fundamental proteger los canales de comunicación y asegurarse de que solo los productores y consumidores autorizados puedan generar y procesar eventos. Además, se deben implementar medidas para evitar la inyección de eventos maliciosos que puedan comprometer el sistema.
* **Rendimiento**: El rendimiento es otro desafío en las EDA, especialmente cuando se manejan sistemas de alta concurrencia o se procesan grandes cantidades de eventos simultáneamente. Las soluciones deben estar optimizadas para garantizar que los eventos se procesen de manera eficiente sin generar cuellos de botella en el sistema.

### Relación con el Internet de las Cosas (IoT):

El artículo también explora la **relación de las EDA con el Internet de las Cosas (IoT)**. Las arquitecturas orientadas a eventos son ideales para manejar dispositivos IoT, ya que estos dispositivos generan constantemente eventos en función de sus interacciones con el entorno. Las EDA pueden procesar estos eventos de manera asíncrona y en tiempo real, lo que permite a los sistemas IoT monitorear y reaccionar ante cambios de estado de manera eficiente.

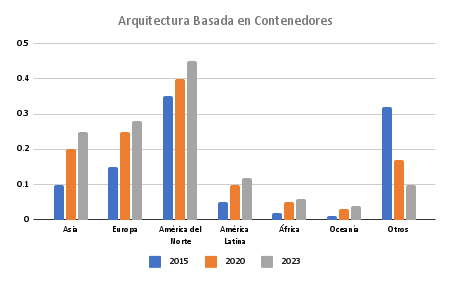
Por ejemplo, en una red de sensores IoT, cada sensor puede actuar como un productor de eventos, enviando datos sobre el entorno, como la temperatura o la humedad. Estos eventos son procesados por una EDA, que puede notificar cambios importantes en los parámetros monitoreados o activar respuestas automáticas, como encender un sistema de climatización si la temperatura alcanza cierto umbral.

### Conclusión:

El artículo concluye que las **Arquitecturas Orientadas a Eventos** ofrecen una solución efectiva para sistemas que requieren una **respuesta rápida y en tiempo real** a los cambios en el entorno. Su capacidad para procesar eventos de manera asíncrona las hace especialmente adecuadas para aplicaciones de IoT y otras áreas donde la escalabilidad y el rendimiento son críticos.

No obstante, implementar una EDA con éxito requiere abordar cuidadosamente los desafíos relacionados con la seguridad y el rendimiento, asegurando que los sistemas puedan manejar la carga de eventos y protegerse contra posibles amenazas. En general, las EDA representan una evolución clave en el diseño de sistemas distribuidos, permitiendo un enfoque más dinámico y flexible en la gestión de eventos.

**Arquitectura Basada en Contenedores**

****

**14**

El artículo se centra en la Arquitectura Basada en Contenedores, describiendo su diseño e implementación para el desarrollo y despliegue de aplicaciones modernas. Esta arquitectura utiliza la tecnología de contenedores para encapsular aplicaciones y sus dependencias, facilitando la portabilidad, escalabilidad y administración eficiente de los entornos de desarrollo y producción.

### Contenedores y Automatización:

La arquitectura basada en contenedores se apoya en herramientas como **Docker**, que permite crear entornos de ejecución estandarizados y portátiles. Un contenedor encapsula la aplicación junto con todas sus dependencias, garantizando que se ejecute de manera consistente en cualquier entorno, ya sea localmente, en la nube o en servidores de producción. Esto soluciona problemas comunes de compatibilidad y configuración que se presentan en arquitecturas tradicionales.

El artículo también explora la **automatización** de tareas mediante **shell scripts** y servicios externos. La automatización es crucial para mejorar la eficiencia del desarrollo y despliegue de aplicaciones, reduciendo el tiempo de configuración manual y minimizando errores humanos. Estos scripts permiten la automatización de procesos repetitivos, como la creación de contenedores, la configuración del entorno, y el despliegue de la aplicación.

### Entorno Local y Despliegue en la Nube:

Uno de los aspectos clave de la arquitectura basada en contenedores es la facilidad para configurar y ejecutar aplicaciones en un **entorno local**, permitiendo a los desarrolladores replicar fácilmente el entorno de producción en sus máquinas. El artículo detalla el proceso de configuración inicial, desde la creación de los contenedores hasta la configuración de redes y volúmenes para persistencia de datos.

Posteriormente, se analiza el **despliegue en la nube**, donde la arquitectura de contenedores se integra con servicios de infraestructura en la nube, permitiendo escalar las aplicaciones según sea necesario. El uso de herramientas de contenedorización, junto con plataformas de orquestación como **Kubernetes**, facilita la administración de múltiples contenedores y la escalabilidad horizontal de los servicios.

### Integración Continua (CI) y Despliegue Continuo (CD):

El artículo también resalta la importancia de la **Integración Continua (CI)** y el **Despliegue Continuo (CD)** para mantener un flujo de desarrollo ágil y eficiente. Estas prácticas se integran en la arquitectura basada en contenedores utilizando herramientas como **Jenkins**, que automatiza la integración de código y la ejecución de pruebas, asegurando que cualquier cambio en el código se pueda verificar y desplegar rápidamente en el entorno de producción.

* **CI**: La Integración Continua implica la integración frecuente del código en un repositorio compartido, seguido de la automatización de la construcción y las pruebas. Esto asegura que el código sea siempre funcional y que los problemas se detecten temprano.
* **CD**: El Despliegue Continuo se basa en la entrega automática de las nuevas versiones de la aplicación, listas para ser desplegadas en producción. Esto permite lanzamientos rápidos y frecuentes, reduciendo el tiempo entre el desarrollo y la entrega de nuevas características o correcciones.

### Dockerfile e Infraestructura:

El artículo aborda los aspectos técnicos relacionados con la creación de **Dockerfiles**, que definen cómo se construyen las imágenes de los contenedores. El diseño de Dockerfiles eficientes es crucial para asegurar que las imágenes sean ligeras, seguras y optimizadas para el rendimiento.

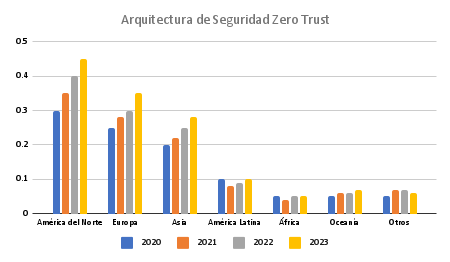
Además, se discuten las **decisiones de diseño** relacionadas con la infraestructura necesaria para garantizar que el despliegue de las aplicaciones sea seguro, escalable y altamente disponible. Esto incluye la elección de plataformas de orquestación, la configuración de redes y la implementación de medidas de seguridad para proteger los contenedores en producción.

### Conclusión:

El artículo concluye que la **arquitectura basada en contenedores** ofrece una solución sólida para el desarrollo y despliegue de aplicaciones modernas, permitiendo a las organizaciones aprovechar al máximo la escalabilidad y flexibilidad de los entornos de nube. La integración de contenedores con herramientas de automatización y CI/CD, como Jenkins, mejora significativamente la eficiencia operativa y la calidad del software.

Sin embargo, el éxito de esta arquitectura depende de un diseño cuidadoso y la correcta configuración de los contenedores y la infraestructura subyacente, asegurando que los sistemas sean seguros, escalables y fáciles de gestionar.

**Arquitectura de Seguridad Zero Trust**

****

**15**

La tesis escrita por Christopher Junior Pesántez Ávila aborda la Arquitectura de Seguridad Zero Trust y su implementación en la infraestructura tecnológica de la empresa SEÑAL X. Este modelo de seguridad tiene como objetivo proteger la información empresarial en un contexto donde las amenazas cibernéticas son cada vez más sofisticadas y las redes tradicionales no pueden garantizar la seguridad completa de los sistemas.

### Introducción a Zero Trust:

La **Arquitectura de Seguridad Zero Trust** es un enfoque de ciberseguridad basado en la premisa de que **no se debe confiar en nadie dentro o fuera de la red** sin una verificación exhaustiva. A diferencia de los modelos de seguridad tradicionales, que presumen que todo lo que está dentro de la red empresarial es confiable, Zero Trust asume que las amenazas pueden surgir desde cualquier punto, por lo que exige la autenticación, autorización y validación continua de todos los dispositivos y usuarios que intenten acceder a los recursos de la organización.

### Implementación en SEÑAL X:

El documento detalla el proceso de implementación de este modelo en la empresa SEÑAL X. La arquitectura Zero Trust se centra en varios principios clave para proteger los activos de la organización:

1. **Verificación continua**: Cada acceso a los sistemas, independientemente de su origen, debe ser autenticado y autorizado. Esto incluye la autenticación multifactor (MFA), que agrega capas adicionales de seguridad al requerir más de un método de verificación de identidad.
2. **Segmentación de la red**: Se implementa la segmentación de la red para minimizar el riesgo de que una brecha comprometa todos los sistemas. Al dividir la red en zonas más pequeñas y controladas, se limita el acceso entre diferentes partes del sistema, reduciendo el impacto de un posible ataque.
3. **Mínimo privilegio**: El modelo de Zero Trust se basa en el principio del **mínimo privilegio**, donde cada usuario o sistema solo tiene el acceso necesario para realizar su función específica, limitando las posibles acciones que un intruso podría realizar en caso de un ataque exitoso.
4. **Monitoreo y análisis en tiempo real**: La supervisión continua de la actividad en la red es fundamental en un entorno Zero Trust. SEÑAL X implementó herramientas de monitoreo que analizan el comportamiento de usuarios y sistemas en tiempo real, permitiendo identificar y responder rápidamente a actividades sospechosas.

### Desafíos y Beneficios:

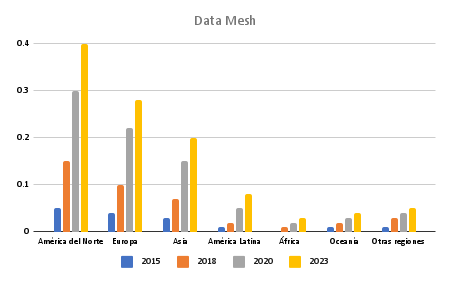
El documento también aborda los desafíos enfrentados durante la implementación de la arquitectura Zero Trust. Uno de los principales obstáculos fue la **adaptación de la infraestructura tecnológica** existente para cumplir con los requisitos de Zero Trust. Esto implicó una inversión significativa en nuevas herramientas de seguridad, así como en la formación del personal para que comprendieran y aplicaran los principios del modelo.

A pesar de estos desafíos, los beneficios observados tras la implementación incluyen una **mayor resiliencia frente a ciberataques** y una **mejor gestión de riesgos** en toda la organización. La capacidad de prevenir movimientos laterales dentro de la red y la protección mejorada de datos sensibles son algunos de los aspectos destacados que mejoraron la postura de seguridad de SEÑAL X.

### Conclusión:

La tesis concluye que la implementación del modelo Zero Trust en SEÑAL X ha fortalecido significativamente la seguridad de la empresa, adaptándola a un entorno digital moderno en el que las amenazas son cada vez más avanzadas. Aunque la transición hacia una arquitectura Zero Trust requiere una planificación meticulosa y una actualización de la infraestructura, los resultados obtenidos en términos de protección de la información y reducción de riesgos justifican la adopción de este enfoque en organizaciones que desean mejorar su ciberseguridad de manera efectiva.

**Data Mesh**

****

**16**

El artículo en la sección 2.4.1 aborda el concepto de **Data Mesh**, un enfoque innovador para la gestión y compartición de datos en organizaciones complejas. A diferencia de los enfoques tradicionales centralizados, Data Mesh distribuye la responsabilidad de los datos a través de diferentes equipos, tratándolos como productos independientes. A continuación, se detallan los principales aspectos de Data Mesh según el artículo:

### Definición de Data Mesh:

**Data Mesh** es una arquitectura que se basa en la descentralización y la federación de datos, proponiendo un cambio significativo en cómo las organizaciones gestionan y comparten sus datos. En lugar de centralizar todos los datos en un único almacén, Data Mesh asigna la responsabilidad de los conjuntos de datos a equipos específicos, cada uno gestionando sus datos como si fueran **productos**. Este enfoque busca superar los desafíos de escalabilidad y mantenimiento asociados con los enfoques tradicionales.

### Principales Servicios de Data Mesh:

1. Creación de Productos de Datos:
   * **Responsabilidad**: En un Data Mesh, los equipos son responsables de la creación y el mantenimiento de sus propios productos de datos. Esto implica definir claramente los datos que se producen, sus formatos, y cómo se deben consumir.
   * **Calidad**: Los equipos deben garantizar la calidad y la relevancia de sus datos, asegurando que sean útiles y estén bien documentados para otros equipos que los consuman.
2. Cambio de Roles:
   * **Descentralización**: Data Mesh promueve una descentralización de la gestión de datos, donde el rol de los equipos cambia de consumidores pasivos a **productores activos** de datos. Esto fomenta una cultura de responsabilidad compartida y colaboración entre equipos.
   * **Especialización**: Cada equipo se convierte en un experto en su conjunto de datos específico, lo que puede mejorar la calidad de los datos y la eficiencia en su uso.
3. Acceso a Datos:
   * **Autonomía**: Los equipos tienen autonomía para acceder y consumir datos de otros equipos dentro del marco de Data Mesh. Esto se logra mediante interfaces estandarizadas y acuerdos de interoperabilidad.
   * **Seguridad y Gobernanza**: Aunque los datos están distribuidos, se deben establecer políticas y mecanismos para asegurar el acceso seguro y la gobernanza de los datos, garantizando que se cumplan las normativas y estándares organizacionales.
4. Infraestructura de Autoservicio:
   * **Autonomía Operacional**: Data Mesh incluye el desarrollo de una infraestructura de autoservicio que permite a los equipos gestionar sus propios datos de manera independiente. Esto incluye herramientas y plataformas que facilitan la creación, almacenamiento, y acceso a los datos.
   * **Soporte Técnico**: La infraestructura debe ser soportada por servicios técnicos que ayuden a los equipos a implementar y mantener sus productos de datos, asegurando que se cumplan los requisitos técnicos y operacionales.

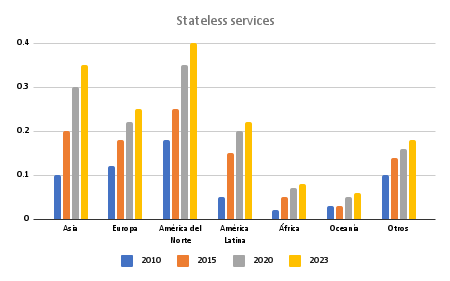
### Beneficios del Data Mesh:

* **Escalabilidad**: Al distribuir la gestión de datos, Data Mesh permite que las organizaciones escalen sus capacidades de manejo de datos sin los cuellos de botella asociados con los enfoques centralizados.
* **Responsabilidad y Calidad**: La asignación de responsabilidad a equipos específicos mejora la calidad de los datos y facilita una mayor alineación con las necesidades del negocio.
* **Flexibilidad y Agilidad**: Data Mesh fomenta una mayor flexibilidad y agilidad al permitir que los equipos gestionen y evolucionen sus datos de manera independiente.

### Conclusión:

El artículo concluye que el enfoque **Data Mesh** representa una evolución significativa en la gestión de datos, particularmente en organizaciones grandes y complejas donde los enfoques tradicionales pueden resultar ineficaces. Al tratar los datos como productos y descentralizar su gestión, Data Mesh ofrece una solución escalable y flexible que puede mejorar la calidad y la accesibilidad de los datos en toda la organización.

**Stateless services**

****

**17**

El artículo examina el concepto de **servicios sin estado (stateless services)** dentro del marco del desarrollo de soluciones basadas en microservicios, destacando su importancia y las implicaciones en la arquitectura de sistemas distribuidos. Aquí están los puntos clave tratados en el artículo:

### Concepto de Servicios Sin Estado:

Servicios sin estado se refieren a servicios que no mantienen información sobre el estado entre solicitudes. En lugar de guardar datos sobre el estado de una sesión o transacción, estos servicios delegan toda la funcionalidad de persistencia a servicios externos. Este enfoque es fundamental en las arquitecturas de microservicios por varias razones:

* **Escalabilidad**: Al no mantener el estado, los servicios sin estado pueden ser escalados horizontalmente de manera más eficiente. Esto significa que se pueden agregar o eliminar instancias de un servicio sin preocuparse por sincronizar el estado entre ellas.
* **Simplicidad**: La ausencia de estado reduce la complejidad en la gestión del servicio, ya que no es necesario manejar la persistencia de datos a nivel del servicio individual.

### Delegación de Persistencia:

Dado que los servicios sin estado no manejan la persistencia, esta debe ser gestionada por servicios externos. Esto suele implicar el uso de **bases de datos** u otros mecanismos de almacenamiento proporcionados por proveedores de nube. Estos recursos externos deben ser cuidadosamente configurados para garantizar el acceso y la comunicación adecuados:

* **Configuración de Red**: La configuración de la red es crucial para asegurar que los servicios sin estado puedan comunicarse de manera eficiente con los servicios de persistencia. Esto incluye la gestión de redes, políticas de acceso y seguridad para asegurar que los datos sean accesibles cuando se necesiten.

### Uso de Servidor Proxy:

El artículo también destaca la importancia de los patrones de servidor proxy en la gestión de servicios sin estado:

* **Enrutamiento Dinámico**: Herramientas como **Netflix Zuul** se utilizan para gestionar el enrutamiento dinámico a los microservicios. Zuul actúa como un punto de entrada único para las solicitudes, dirigiéndolas al microservicio correspondiente basado en la configuración y la lógica definida.
* **Optimización de Solicitudes**: El uso de un servidor proxy ayuda a optimizar las solicitudes y respuestas, reduciendo la latencia y mejorando la eficiencia de las interacciones entre servicios.
* **Centralización de Seguridad**: Un servidor proxy puede centralizar la seguridad de las interacciones públicas, manejando la autenticación, autorización y la protección contra ataques, lo cual es fundamental en un entorno distribuido.

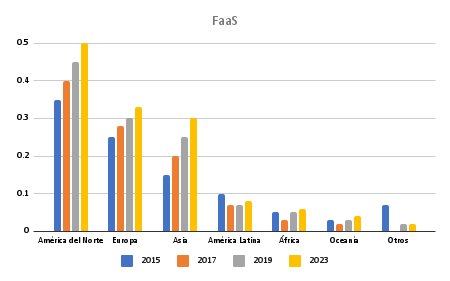
### Beneficios de los Servicios Sin Estado:

1. **Flexibilidad y Escalabilidad**: La naturaleza sin estado permite una mayor flexibilidad y facilidad para escalar servicios de manera horizontal, mejorando la capacidad de respuesta y la disponibilidad del sistema.
2. **Resiliencia y Recuperación**: Los servicios sin estado pueden ser más resilientes, ya que cualquier instancia puede ser reemplazada sin afectar el estado global del sistema. Esto facilita la recuperación ante fallos y la actualización de servicios.
3. **Simplificación del Desarrollo**: Al delegar la persistencia a servicios externos, los desarrolladores pueden centrarse en la lógica de negocio de los servicios sin estado, reduciendo la complejidad del código y la necesidad de manejar la persistencia interna.

### Conclusión:

El artículo concluye que diseñar **servicios sin estado** es una práctica esencial en el desarrollo de arquitecturas basadas en microservicios, ofreciendo beneficios significativos en términos de escalabilidad, flexibilidad y eficiencia. La delegación de la persistencia a servicios externos y el uso de patrones de servidor proxy son componentes clave para la implementación exitosa de estas arquitecturas en soluciones modernas.

**FaaS**

****

**18**

El artículo explora el concepto de **Funciones como Servicio (FaaS)**, un modelo de computación en la nube también conocido como **"serverless"** o sin servidor. Este enfoque permite ejecutar funciones individuales en respuesta a eventos específicos, sin que los desarrolladores necesiten gestionar los servidores subyacentes. Aquí se detallan los puntos clave abordados en el artículo:

### Concepto de FaaS:

**Funciones como Servicio (FaaS)** es un modelo de computación en la nube que se basa en la ejecución de funciones individuales en respuesta a eventos. A diferencia de las arquitecturas tradicionales basadas en servidores, FaaS permite a los desarrolladores centrarse en el código de las funciones, mientras que la infraestructura subyacente es gestionada automáticamente por la plataforma de nube. Esto significa que:

* **Ejecución por Evento**: Las funciones se ejecutan en respuesta a eventos específicos, como solicitudes HTTP, cambios en bases de datos, o mensajes en colas. Una vez que se completa la ejecución del evento, la plataforma puede cerrar el entorno de ejecución, optimizando así los recursos.
* **Independencia de Funciones**: Las funciones en una arquitectura FaaS son independientes y modulares, permitiendo que cada función sea desarrollada, desplegada y corregida de manera aislada. Esto favorece una mayor flexibilidad y simplicidad en el desarrollo.

### Implementación de FaaS:

El artículo destaca la implementación de FaaS utilizando varias tecnologías, entre ellas **Azure Functions**. Azure Functions es una plataforma popular para FaaS que se integra con otras herramientas y servicios de Microsoft, facilitando la creación y gestión de funciones en la nube.

* **Azure Functions**: La elección de Azure Functions se justifica por su capacidad de integrarse con el ecosistema de Microsoft, proporcionando una plataforma robusta para ejecutar funciones sin servidor y aprovechar otros servicios de Azure.
* **Alternativas a Azure Functions**: El artículo también explora **OpenFaaS** como una plataforma alternativa. OpenFaaS es una solución de código abierto que permite ejecutar funciones en Kubernetes y otras infraestructuras, proporcionando una opción flexible y extensible para la implementación de FaaS.

### Beneficios de FaaS:

1. **Reducción de la Gestión de Infraestructura**: Al liberar a los desarrolladores de la necesidad de gestionar servidores, FaaS permite un enfoque más centrado en el desarrollo del código y la lógica de negocio.
2. **Escalabilidad Automática**: Las plataformas FaaS gestionan automáticamente la escalabilidad, ajustando la capacidad de ejecución de funciones según la demanda del evento.
3. **Costos Basados en el Uso**: En lugar de pagar por recursos de servidor constantes, los costos en un modelo FaaS se basan en el uso real de las funciones, lo que puede resultar en una reducción de costos.

### Consideraciones y Desafíos:

Aunque FaaS ofrece numerosos beneficios, también presenta algunos desafíos:

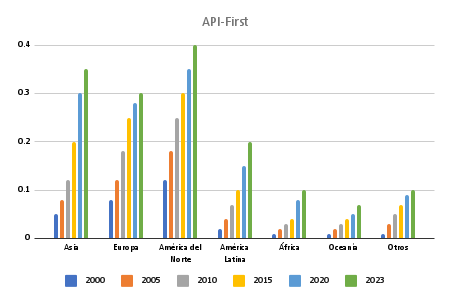
* **Limitaciones de Ejecución**: Las funciones pueden tener limitaciones en cuanto a tiempo de ejecución, memoria y recursos, lo que debe ser considerado al diseñar la arquitectura.
* **Gestión de Estado**: Dado que las funciones son efímeras, el manejo del estado y la persistencia de datos debe ser gestionado a través de servicios externos, lo que puede añadir complejidad adicional.

### Conclusión:

El artículo concluye que el modelo **FaaS** proporciona una solución eficaz para ejecutar funciones en la nube sin la necesidad de gestionar la infraestructura subyacente. Al centrarse en el código y permitir la ejecución basada en eventos, FaaS mejora la flexibilidad y la escalabilidad de las aplicaciones. La comparación entre plataformas como Azure Functions y OpenFaaS subraya la importancia de seleccionar la herramienta adecuada en función de los requisitos específicos del proyecto y el entorno tecnológico.

En resumen, **FaaS** ofrece una alternativa poderosa y eficiente para el desarrollo de aplicaciones basadas en eventos, permitiendo a los desarrolladores concentrarse en la creación de funcionalidades mientras se benefician de la gestión automática de la infraestructura.

**API-First**

****

**19**

El artículo analiza el enfoque API-First como un método de desarrollo emergente en el que las APIs (Interfaces de Programación de Aplicaciones) se sitúan en el centro del diseño de software. Esto significa que, en lugar de desarrollar primero las funcionalidades internas y luego crear APIs para exponerlas, los equipos de desarrollo empiezan definiendo y diseñando las APIs antes de trabajar en el resto de la aplicación. Esta metodología facilita que todos los componentes del sistema (front-end, back-end y otros servicios) trabajen en torno a una API bien definida desde el principio.

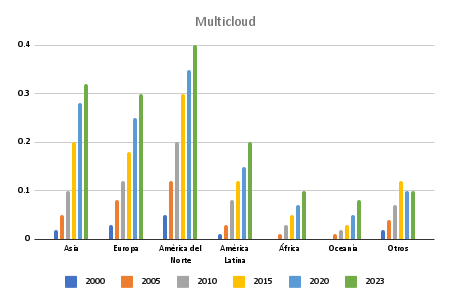
Una de las principales ventajas del enfoque API-First es que permite a los equipos de desarrollo trabajar de manera más colaborativa y paralela, reduciendo la dependencia entre los distintos equipos y mejorando la eficiencia. Por ejemplo, mientras el equipo de back-end trabaja en la implementación de las APIs, el equipo de front-end puede empezar a desarrollar sus funcionalidades utilizando mockups o especificaciones de la API, sin tener que esperar a que el back-end esté completamente terminado. Este enfoque también promueve la consistencia en el diseño de las APIs, lo que resulta en una mejor experiencia de usuario y desarrollador.

El artículo también resalta que, aunque el enfoque API-First está cada vez más adoptado en la industria, aún no ha sido suficientemente estudiado en la literatura académica, lo que podría indicar la necesidad de más investigación en este campo. Sin embargo, a través de un caso de estudio presentado en el artículo, se observa que la implementación de este enfoque en la práctica tiene desafíos importantes, especialmente en relación con la dependencia de herramientas y tecnologías. Las APIs diseñadas inicialmente a menudo requieren ajustes durante el proceso de desarrollo, lo que puede introducir dificultades para mantener las especificaciones actualizadas y garantizar la coherencia entre la definición y la implementación de la API.

Entre los retos mencionados, se encuentran las limitaciones en la automatización del proceso de implementación y las dificultades para garantizar una experiencia fluida al usar las herramientas de diseño de API. Esto sugiere que la adopción del enfoque API-First requiere no solo un cambio en la mentalidad de desarrollo, sino también una infraestructura de herramientas adecuada para que las APIs puedan ser diseñadas, documentadas y mantenidas eficientemente.

En resumen, el artículo destaca la importancia del enfoque API-First en el desarrollo de software moderno, subrayando los beneficios de colocar las APIs en el núcleo del proceso de diseño. Sin embargo, también pone de relieve las consideraciones prácticas y los desafíos técnicos que deben ser abordados para que este enfoque sea adoptado de manera efectiva, incluyendo la necesidad de herramientas robustas y procesos bien definidos para garantizar una implementación exitosa.

**Multicloud**

****

20

El artículo explora el enfoque **Multicloud**, que se refiere al uso de múltiples servicios de computación en la nube proporcionados por diferentes proveedores. Este enfoque está ganando popularidad entre las empresas debido a su capacidad para mejorar la **gestión de sistemas** y ofrecer una serie de beneficios importantes que serían difíciles de lograr utilizando solo infraestructura propia o confiando en un único proveedor de nube.

Uno de los principales beneficios mencionados es la **alta disponibilidad de servicios**, lo que significa que las empresas pueden minimizar las interrupciones operativas y mejorar la **resiliencia** de sus sistemas al distribuir sus cargas de trabajo entre varios proveedores de nube. Al no depender de un solo proveedor, las organizaciones reducen el riesgo de sufrir caídas de servicio o problemas técnicos asociados con un único punto de fallo. Además, se destaca que este enfoque contribuye a **mejorar la seguridad**, ya que permite aplicar medidas de protección a través de diferentes entornos y así reducir la exposición a vulnerabilidades específicas de una sola plataforma.

Otro beneficio clave es la **escalabilidad**. Con **Multicloud**, las empresas pueden aumentar o reducir la capacidad según sea necesario, ajustando sus recursos de computación de manera eficiente. Esta flexibilidad también tiene un impacto positivo en la **reducción de costos**, ya que permite a las organizaciones optimizar el uso de la infraestructura en función de las necesidades y características de cada proveedor.

El artículo subraya la importancia de seguir buenas prácticas para la implementación de plataformas **Multicloud**. Esto incluye establecer configuraciones y políticas de seguridad **consistentes** entre las diferentes nubes, de manera que los datos sensibles puedan clasificarse y protegerse adecuadamente en cada entorno. La **administración** y **monitoreo multiplataforma** también son esenciales, ya que permiten a las empresas mantener un control centralizado sobre sus operaciones en varias nubes y garantizar que se cumplan los niveles de rendimiento y seguridad requeridos.

A pesar de los beneficios, el enfoque **Multicloud** no está exento de **riesgos**. El artículo menciona desafíos relacionados con la **complejidad** de administrar múltiples entornos de nube, la necesidad de personal capacitado, y los riesgos de **compatibilidad** entre plataformas. Sin embargo, estos riesgos son manejables si se implementan controles adecuados, como el uso de herramientas especializadas para monitorear y gestionar recursos distribuidos entre diferentes nubes.

Finalmente, uno de los puntos más destacados es la capacidad del enfoque **Multicloud** para evitar el **bloqueo de proveedor** (vendor lock-in). Al utilizar varios proveedores de nube, las organizaciones pueden evitar quedar atrapadas en la dependencia de un único proveedor, lo que les otorga mayor flexibilidad y libertad para elegir la solución más adecuada a sus necesidades a lo largo del tiempo. Además, esto les permite aprovechar las mejores características de cada plataforma en lugar de limitarse a las capacidades de un solo proveedor.

En resumen, el artículo concluye que la adopción de un enfoque **Multicloud** ofrece a las empresas una estrategia sólida para mejorar la **eficiencia**, la **seguridad**, y la **flexibilidad** de sus operaciones, optimizando el uso de recursos en múltiples plataformas y reduciendo los riesgos asociados con la dependencia de un solo proveedor de servicios en la nube.

**Bibliografía**

**Microservicios:**

Datos de los graficos: <https://www.jetbrains.com/lp/devecosystem-2023/>

Articulo de referencia: <http://138.59.13.30/bitstream/10786/1277/1/93%20Arquitectura%20de%20Software%20basada%20en%20Microservicios%20para%20Desarrollo%20de%20Aplicaciones%20Web.pdf>

Datos de los gráficos:

**Arquitectura de software**

Artículo

<http://scielo.sld.cu/scielo.php?pid=S2227-18992019000100143&script=sci_arttext>

Datos de los gráficos:

**Monolítica**

Artículo

<https://www.researchgate.net/profile/Anabella-Guimarey/publication/348309479_Beneficios_y_riesgos_de_migrar_una_arquitectura_monolitica_a_microservicios/links/5ff749b0a6fdccdcb83b3e20/Beneficios-y-riesgos-de-migrar-una-arquitectura-monolitica-a-microservicios.pdf>

Datos de los gráficos:

<https://nix-united.com/blog/monolith-vs-microservices-how-to-choose-the-right-architecture/>

<https://www.iopex.com/blogs/moving-from-monoliths-to-microservices-through-application-modernization/>

**Hexagonal**

Artículo

<https://riunet.upv.es/handle/10251/198667>

Datos de los gráficos:

<https://0x5.uk/2023/09/28/what-is-hexagonal-architecture/>

<https://www.nirmitee.io/blogs/hexagonal-architecture-what-you-get-out-of-it>

**Event-Sourcing**

Artículo

<https://openaccess.uoc.edu/handle/10609/132648>

Datos de los gráficos:

<https://estuary.dev/event-driven-vs-event-sourcing/>

<https://www.xolv.io/blog/articles/event-sourcing-the-future-proof-design-pattern-for-2023-and-beyond/>

**RESTful**

Artículo

<https://repositorio.uan.edu.co/items/c776c5e1-c555-4801-8b18-715fa9abcc96>

<https://www.redhat.com/architect/pros-and-cons-event-sourcing-architecture-pattern>

Datos de los gráficos:

[**https://www.postman.com/state-of-api/api-global-growth/**](https://www.postman.com/state-of-api/api-global-growth/)

[**https://nordicapis.com/top-architectural-styles-for-apis-in-2023/**](https://nordicapis.com/top-architectural-styles-for-apis-in-2023/)

[**https://www.postman.com/state-of-api/api-technologies/**](https://www.postman.com/state-of-api/api-technologies/)

[**https://www.decisionanalytic.com/post/exploring-the-top-api-architecture-styles-of-2023-rest-graphql-and-beyond**](https://www.decisionanalytic.com/post/exploring-the-top-api-architecture-styles-of-2023-rest-graphql-and-beyond)

[**https://nordicapis.com/20-impressive-api-economy-statistics/**](https://nordicapis.com/20-impressive-api-economy-statistics/)

**GraphQL**

Artículo

<https://repositorio.utn.edu.ec/handle/123456789/13293>

Datos de los gráficos:

**API Gateway**

Artículo

<https://repositorio.uss.edu.pe/handle/20.500.12802/12164>

Datos de los gráficos:

[**https://scalac.io/blog/grpc-vs-rest-vs-graphql/**](https://scalac.io/blog/grpc-vs-rest-vs-graphql/)

[**https://www.moesif.com/blog/api-analytics/api-strategy/Graphql-vs-Rest-A-Comprehensive-Comparison/**](https://www.moesif.com/blog/api-analytics/api-strategy/Graphql-vs-Rest-A-Comprehensive-Comparison/)

[**https://wundergraph.com/blog/graphql\_rest\_openapi\_trend\_analysis\_2023**](https://wundergraph.com/blog/graphql_rest_openapi_trend_analysis_2023)

[**https://www.ibm.com/think/insights/seven-key-insights-on-graphql-trends**](https://www.ibm.com/think/insights/seven-key-insights-on-graphql-trends)

[**https://www.decisionanalytic.com/post/exploring-the-top-api-architecture-styles-of-2023-rest-graphql-and-beyond**](https://www.decisionanalytic.com/post/exploring-the-top-api-architecture-styles-of-2023-rest-graphql-and-beyond)

**Pruebas Unitarias**

Artículo

<https://repositorio.uci.cu/bitstream/123456789/7676/1/TD_08661_16.pdf>

Datos de los gráficos:

**Pruebas de Integración**

Artículo

<https://repositorio.uci.cu/bitstream/ident/TD_03612_10/1/TD_03612_10.pdf>

Datos de los gráficos:

**Balanceo de Carga**

Artículo

<https://repositorio.ucenfotec.ac.cr/handle/123456789/503>

Datos de los gráficos:

**Caching**

Artículo

<http://repositoriodspace.unipamplona.edu.co/jspui/handle/20.500.12744/5596>

Datos de los gráficos:

**Clustering**

Artículo

<https://repositorio.uci.cu/bitstream/ident/TD_02825_10/1/TD_02825_10.pdf>

Datos de los gráficos:

**Desarrollo basado en eventos**

Artículo

<https://openaccess.uoc.edu/handle/10609/132648>

Datos de los gráficos:

**Infraestructura como Código (IaC)**·

Artículo

<https://oa.upm.es/68217/>

Datos de los gráficos:

**Arquitectura Serverless**

Artículo

<https://riunet.upv.es/handle/10251/124989>

Datos de los gráficos:

<https://www.gminsights.com/industry-analysis/serverless-architecture-market>

<https://www.infoq.com/news/2023/09/state-serverless-report/>

<https://www.marketsandmarkets.com/Market-Reports/serverless-architecture-market-64917099.html>

<https://futuremarketanalytics.com/report/serverless-architecture-market/>

<https://www.sitech.me/blog/5-cloud-computing-trends-that-will-shape-2023>

**Seguridad de Software**

Artículo

<https://dialnet.unirioja.es/servlet/articulo?codigo=9435897>

Datos de los gráficos:

[**https://cyberpandit.org/global-cyber-security-landscape-2023/**](https://cyberpandit.org/global-cyber-security-landscape-2023/)

[**https://www.varonis.com/blog**](https://www.varonis.com/blog)

[**https://www.crowdstrike.com/resources/reports/crowdstrike-2023-global-threat-report/**](https://www.crowdstrike.com/resources/reports/crowdstrike-2023-global-threat-report/)

**Desarrollo Nativo en la Nube**

Artículo

<https://repositorio.ucp.edu.co/entities/publication/18469d1c-2c37-44a5-a299-41dd79e760f7>

Datos de los gráficos:

<https://cloudnativenow.com/features/2023-marks-the-rise-of-cloud-native-platforms/>

<https://mogenius.com/blog-posts/the-state-of-cloud-native-software-development>

<https://www.cncf.io/reports/cncf-annual-survey-2023/>

**Edge Computing**

Artículo

<https://core.ac.uk/download/pdf/334999297.pdf>

Datos de los gráficos:

**Desarrollo Ágil y DevOps**

Artículo

[https://repositorio.ucv.edu.pe/bitstream/handle/20.500.12692/133838/Altamirano\_TLH-SD.p](https://repositorio.ucv.edu.pe/bitstream/handle/20.500.12692/133838/Altamirano_TLH-SD.pdf?sequence=1&isAllowed=y)

[df?sequence=1&isAllowed=y](https://repositorio.ucv.edu.pe/bitstream/handle/20.500.12692/133838/Altamirano_TLH-SD.pdf?sequence=1&isAllowed=y)

Datos de los gráficos:

<https://digital.ai/press-releases/15th-state-of-agile-report-shows-notable-rise-in-agile-adoption-across-the/>

<https://www.zippia.com/advice/agile-statistics/>

<https://www.scrum.org/resources/2023-devops-predictions-agile-category>

**Micro-Frontends**

Artículo

<https://repository.udistrital.edu.co/items/f758313a-5091-46e6-a12a-71029db65bf2>

Datos de los gráficos:

**Arquitectura Orientada a Eventos (EDA)**

Artículo

<https://oa.upm.es/37339/>

Datos de los gráficos:

**Arquitectura Basada en Contenedores**

Artículo

<https://riuma.uma.es/xmlui/handle/10630/19476>

Datos de los gráficos:

**Arquitectura de Seguridad Zero Trust:**

Artículo

<https://dspace.ups.edu.ec/handle/123456789/25023>

Datos de los gráficos:

**Data Mesh**

Artículo

<https://titula.universidadeuropea.es/handle/20.500.12880/8874>

Datos de los gráficos:

**Stateless Services**

Artículo

<https://riunet.upv.es/handle/10251/170177>

Datos de los gráficos:

**FaaS**

Artículo

<https://oa.upm.es/65192/>

Datos de los gráficos:

**API First**

Artículo

<https://publicaciones.unpa.edu.ar/index.php/ICTUNPA/article/view/1103>

Datos de los gráficos:

<https://www.softwareag.com/en_corporate/blog/top-10-api-trends-2024.html>

<https://www.sensedia.com/post/6-api-trends-for-2023>

<https://www.postman.com/state-of-api/api-first-strategies/>

https://nordicapis.com/11-trends-to-watch-in-the-api-economy-in-2023/

**Multicloud**

Artículo

<http://44.209.83.190/bitstream/handle/20.500.14230/10627/REF-1633553884-1.pdf?sequence=1&isAllowed=y>

Datos de los gráficos:

<https://www.dbta.com/Editorial/News-Flashes/IT-Leaders-Predict-Increased-Multi-Cloud-Adoption-in-2023-156367.aspx>

<https://www.newhorizons.com/resources/blog/multi-cloud-adoption>