
Alcaldía de Medellín
Distrito de
Ciencia, Tecnología e Innovación
SICM
SISTEMA DE INFOMACIÓN CATASTRAL DE MEDELLÍN

Actualización del Sistema de Información Catastral de Medellín: Enfoque en Escalabilidad, Sostenibilidad y Robustez

Secretaría de Gestión y Control Territorial

Subsecretaria de Catastro

Equipo Apoyo a la Información Catastral
Líder de Proyecto Ing. Julio César Mendoza

Este documento aborda la identificación de tecnologías clave para la actualización del sistema de información catastral de Medellín, destacando la infraestructura en la nube, bases de datos distribuidas y modelos de gobernanza. También enfatiza la importancia de la seguridad, interoperabilidad y sostenibilidad en la gestión catastral. Se incluyen fuentes de información relevantes y estrategias para optimizar la toma de decisiones y los recursos en el desarrollo e implementación de estos sistemas.

Versión	Fecha	Autor	Descripción de cambios	Estado	Reviso	Aprobó
1.0	2025-02-24	Ing. Lina Maritza Galviz Ing. Vivian Yépez Ing. Diego Gómez Ing. Johan Bejarano	Documento inicial creado.	Final	Ing. Danny Salcedo	Ing. Julio César Mendoza

1. Contenido

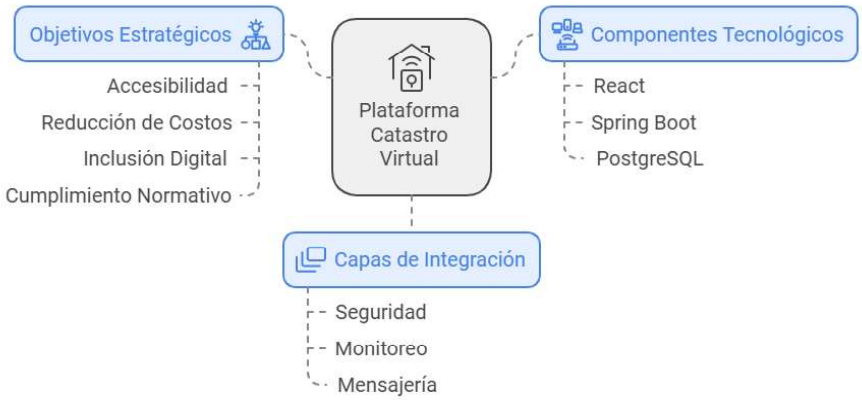
2. Introducción

El presente documento detalla la Arquitectura de Referencia para la plataforma Catastro Virtual, una solución tecnológica diseñada para modernizar y optimizar los servicios catastrales ofrecidos por la Alcaldía de Medellín. Este proyecto, desarrollado bajo la dirección de la Secretaría de Gestión y Control Territorial, tiene como propósito fundamental facilitar a los ciudadanos el acceso a servicios catastrales de manera eficiente, segura y transparente a través de plataformas web y móviles.

En el marco del programa de fortalecimiento del Catastro Distrital, esta arquitectura de referencia establece las bases para una solución robusta, alineada con las políticas de gobierno digital. La plataforma busca resolver las necesidades actuales de interacción entre los ciudadanos y la administración distrital, priorizando la accesibilidad, la reducción de costos operativos, la inclusión digital y el cumplimiento de las normativas de seguridad y privacidad, así como las normativas vigentes que establecen lineamientos específicos de operación, como las resoluciones relacionadas con la reglamentación técnica para la formación, actualización, conservación y difusión catastral. Esto asegurará que la plataforma cumpla con los lineamientos legales y objetivos de transparencia, confiabilidad y sostenibilidad a largo plazo.

Este documento incluye una descripción detallada de los componentes tecnológicos que conforman la solución, destacando el uso de tecnologías modernas como React, React Native, Kong Gateway, Spring Boot, Keycloak, PostgreSQL, Redis, entre otras. Además, se describen las capas de integración, seguridad, monitoreo y mensajería que garantizan el correcto funcionamiento del sistema bajo principios de escalabilidad, alta disponibilidad y resiliencia.

A través de esta arquitectura de referencia, se establecen los lineamientos para el diseño, desarrollo e implementación de la plataforma, asegurando su alineación con los objetivos estratégicos de modernización, eficiencia y sostenibilidad de los servicios públicos de Medellín.



2.1. Propósitos

El propósito del presente documento es definir la Arquitectura de Referencia para la plataforma Catastro Virtual, estableciendo las directrices técnicas y funcionales necesarias para su diseño, desarrollo e implementación. Este documento servirá como guía fundamental para asegurar que

la solución propuesta sea coherente con los objetivos estratégicos de modernización y digitalización de los servicios públicos establecidos por la Alcaldía de Medellín.

La plataforma Catastro Virtual tiene como meta principal transformar la interacción entre los ciudadanos y la administración municipal, facilitando la gestión de trámites catastrales a través de herramientas digitales accesibles y seguras. Para cumplir con este objetivo, el documento detalla los siguientes aspectos clave:

1. Estandarización Tecnológica:
    - Proveer un marco tecnológico basado en mejores prácticas y estándares de la industria, asegurando una solución modular, escalable y sostenible.
    - Garantizar la interoperabilidad entre los diferentes componentes del sistema, promoviendo la reutilización y simplificación en el desarrollo de servicios futuros.
2. Optimización de Procesos:
    - Facilitar la automatización de trámites catastrales, reduciendo la carga operativa manual, los costos asociados y los tiempos de respuesta.
    - Mejorar la experiencia del usuario mediante interfaces intuitivas y funcionalidades alineadas con las necesidades de la ciudadanía.
3. Transparencia y Seguridad:
    - Establecer mecanismos de seguridad robustos para proteger los datos personales y transaccionales de los usuarios, cumpliendo con las normativas de privacidad aplicables.
    - Fomentar la transparencia mediante el acceso en tiempo real a información actualizada sobre el estado de los trámites y la disponibilidad de datos catastrales.
4. Alineación Estratégica:
    - Contribuir al fortalecimiento del Catastro Distrital, en línea con el programa de gobierno digital y las políticas de modernización pública promovidas a nivel local, nacional e internacional.

Este documento no solo busca proporcionar una guía técnica para el equipo de desarrollo e implementación, sino también servir como una herramienta de referencia para las diferentes partes interesadas, garantizando que todos los actores involucrados compartan una visión clara y unificada del alcance y los objetivos del proyecto.

2.2. Audiencia

Este documento está dirigido a todas las partes interesadas en el diseño, desarrollo e implementación de la Arquitectura de Referencia para la plataforma Catastro Virtual. La audiencia se compone de los siguientes perfiles clave:

1. Equipo de Desarrollo Técnico:
    - Arquitectos de Software: Responsables de diseñar y validar la arquitectura tecnológica, asegurando que cumpla con los requisitos funcionales y no funcionales del sistema.
    - Desarrolladores de Software: Encargados de implementar los diferentes componentes de la plataforma, incluyendo el backend, frontend y servicios de integración.
    - Ingenieros de DevOps: Responsables de la configuración, despliegue, monitoreo y mantenimiento de los entornos de desarrollo, pruebas y producción.
2. Entidades Administrativas:
    - Secretaría de Gestión y Control Territorial: Principal promotora del proyecto y encargada de supervisar que la solución cumpla con los objetivos estratégicos del Catastro Distrital.
    - Subsecretaría de Catastro: Usuario funcional clave, quien garantizará que los requerimientos técnicos y operativos estén alineados con las necesidades catastrales de la ciudad.
3. Equipo de Seguridad Informática:
    - Especialistas en Seguridad de la Información: Encargados de garantizar la protección de los datos y el cumplimiento de normativas de privacidad y ciberseguridad.
4. Stakeholders Estratégicos:

    - Líderes de Proyecto y Directivos: Quienes necesitan comprender los lineamientos técnicos y estratégicos de la arquitectura para tomar decisiones informadas y alinear los recursos necesarios para el éxito del proyecto.
5. Usuarios Finales y Ciudadanos (de forma indirecta):
    - Aunque este documento no está diseñado directamente para ellos, sus necesidades y expectativas son la base para la construcción de una plataforma accesible, segura y eficiente.

2.3. Propósito de la Audiencia

Cada grupo identificado tendrá acceso a este documento con el objetivo de:

- Comprender el alcance y los componentes clave de la arquitectura.
- Alinear esfuerzos y responsabilidades según los roles específicos.
- Garantizar que la implementación de la plataforma cumpla con los objetivos funcionales y técnicos establecidos.

3. Justificación De Proyecto

La implementación de una plataforma digital segura para la gestión de servicios distritales en Medellín, accesible a través de una aplicación móvil y un portal web, responde a la necesidad de modernizar y optimizar los procesos administrativos, mejorando la interacción entre la ciudadanía y la administración municipal. A continuación se detallan las razones y beneficios clave que justifican este proyecto:

1. **Mejora en la Eficiencia y Reducción de Costos**

   - Automatización de Procesos: La digitalización de trámites y servicios permitirá automatizar tareas rutinarias, reduciendo la carga de trabajo manual y minimizando errores administrativos. Esto resultará en una gestión más eficiente y rápida de las solicitudes ciudadanas.

   - Ahorro de Recursos: Al reducir la dependencia de procesos en papel y la necesidad de presencia física en oficinas, se disminuirán significativamente los costos operativos y se optimizarán los recursos humanos y materiales disponibles.

2. Accesibilidad y Conveniencia para los Ciudadanos

   - Acceso Remoto: Los ciudadanos podrán acceder a los servicios y realizar trámites en cualquier momento y desde cualquier lugar, eliminando barreras geográficas y temporales. Esto es especialmente beneficioso para aquellos con limitaciones de movilidad o residentes en áreas periféricas.
   - Simplificación de Trámites: La plataforma ofrecerá una interfaz intuitiva y amigable que facilitará la realización de trámites, reduciendo la complejidad y mejorando la experiencia del usuario.

3. Transparencia y Participación Ciudadana

   - Información en Tiempo Real: La disponibilidad de información actualizada y accesible sobre el estado de los trámites y actividades municipales mejorará la transparencia y la confianza de los ciudadanos en la administración pública.
   - Fomento de la Participación: La plataforma facilitará la participación activa de los ciudadanos en la toma de decisiones y en la formulación de políticas públicas mediante el acceso a información relevante y la posibilidad de interactuar con grupos de interés.

4. Seguridad y Protección de Datos

   - Protección de Datos Personales: La implementación de medidas de seguridad avanzadas garantizará la protección de los datos personales de los ciudadanos, cumpliendo con las normativas de privacidad y seguridad de la información.

- Confianza en la Plataforma: Al proporcionar un entorno seguro y confiable, se fomentará la confianza de los usuarios en la plataforma, incentivando su uso y adopción.

5. Apoyo a la Política de Gobierno Digital

- Alineación con Estrategias Nacionales e Internacionales: Este proyecto se alinea con la política de gobierno digital, promovida tanto a nivel nacional como internacional, que busca modernizar la administración pública mediante el uso de tecnologías de la información y la comunicación.
- Innovación y Modernización: La plataforma representará un avance significativo hacia la innovación y modernización de los servicios públicos, posicionando a Medellín como una ciudad líder en la adopción de soluciones digitales.

6. Impacto Económico y Social

- Impulso al Desarrollo Económico: La mejora en la eficiencia de los trámites y la accesibilidad a la información geográfica y estadística beneficiará a las empresas y comerciantes locales, impulsando el desarrollo económico de la ciudad.
- Inclusión Digital: La plataforma contribuirá a la inclusión digital de todos los segmentos de la población, cerrando brechas y asegurando que más ciudadanos puedan beneficiarse de las tecnologías modernas.
- Calidad de Vida Mejorada: La facilidad de acceso a servicios y la mejora en la eficiencia administrativa tendrán un impacto positivo en la calidad de vida de los ciudadanos, facilitando la gestión de sus necesidades diarias.

El proyecto de una oficina virtual para la gestión de servicios distritales en Medellín es una iniciativa crucial para modernizar la administración pública, mejorar la eficiencia y transparencia de los trámites, y fomentar la participación ciudadana. La implementación de esta plataforma digital no solo optimizará los recursos y procesos actuales, sino que también posicionará a Medellín como una ciudad innovadora y avanzada en el uso de tecnologías digitales para el bienestar de sus ciudadanos.

4. Objetivo General

Desarrollar e implementar una plataforma digital segura, accesible a través de la aplicación móvil de la entidad en www.medellin.gov.co, que permita a los ciudadanos registrarse, gestionar sus credenciales y acceder a una variedad de servicios y opciones de atención. La plataforma integrará la consulta y emisión de certificados catastrales existentes, solicitudes de cambios de propietario, así como otro tipo de mutaciones, seguimiento del estado de trámites, información geográfica, publicación de notificaciones, radicación de trámites específicos, además de proporcionar estadísticas, noticias de relevancia y enlaces con grupos de interés. Este proyecto se alinea con la política de gobierno digital, promoviendo la transparencia, eficiencia y accesibilidad de los servicios públicos, facilitando la interacción entre los ciudadanos y la administración Distrital mediante el uso de tecnologías digitales avanzadas.

4.1. Objetivos Específicos

4.1.1. Implementar el Registro y Autenticación Segura:

- Desarrollar un sistema de registro y autenticación de usuarios que garantice la seguridad y privacidad de los datos personales de los ciudadanos, cumpliendo con las normativas de protección de datos.

4.1.2. Integrar los Servicios Catastrales:

- Incorporar funcionalidades para la consulta y emisión de certificados catastrales existentes, así como la gestión de cambios de propietario y otro tipo de mutaciones, asegurando la precisión y actualización de la información catastral disponible para los ciudadanos.

4.1.3. Desarrollar Herramientas de Seguimiento y Notificación:

- Crear módulos que permitan a los usuarios consultar el estado de sus trámites en tiempo real, recibir notificaciones automáticas sobre actualizaciones y cambios en sus solicitudes, y acceder a publicaciones oficiales relacionadas.

4.1.4. Proveer Información Geográfica y Estadística:

- Integrar sistemas de información geográfica (GIS) y módulos de análisis estadístico que permitan a los usuarios acceder a mapas interactivos, datos geoespaciales y estadísticas relevantes para sus necesidades e intereses.

4.1.5. Fomentar la Transparencia y Participación Ciudadana:

- Establecer canales de comunicación y enlace con grupos de interés y comunidades, proporcionando noticias de trascendencia y facilitando la participación ciudadana en la toma de decisiones mediante la publicación de información transparente y accesible, en línea con la política de gobierno digital.

5. Definiciones, Acrónimos

5.1. Definiciones

- Cloud Native: Se refiere a una metodología de diseño y desarrollo de aplicaciones que aprovechan al máximo los servicios y características de la computación en la nube. Este enfoque está orientado a crear aplicaciones escalables, resilientes, y fáciles de gestionar en entornos cloud, utilizando principios y prácticas específicas como:
    1. Microservicios: Arquitectura basada en dividir las aplicaciones en pequeños servicios independientes que se comunican entre sí, permitiendo un desarrollo y despliegue más ágiles.
    2. Contenedores: Uso de tecnologías como Docker para encapsular aplicaciones y sus dependencias en entornos aislados y portátiles.
    3. Orquestación y Automatización: Implementación de herramientas como Kubernetes para gestionar el despliegue, escalado y operación de aplicaciones en contenedores.
    4. DevOps: Integración continua y entrega continua (CI/CD) para mejorar la colaboración entre desarrolladores y operaciones, facilitando un ciclo de desarrollo más rápido y fiable.
    5. Infraestructura Inmutable: Configuración de entornos de producción que no cambian después del despliegue inicial, mejorando la estabilidad y previsibilidad.
    6. Escalabilidad y Resiliencia: Diseño de aplicaciones para escalar automáticamente en respuesta a la demanda y para recuperarse rápidamente de fallos.

  Las aplicaciones Cloud Native están diseñadas para aprovechar la flexibilidad y la escalabilidad de la infraestructura en la nube, lo que les permite responder de manera eficiente a cambios en la carga de trabajo y requisitos del negocio.
- API Management: API Management es un enfoque holístico para gestionar todo el ciclo de vida de las APIs, desde su diseño y desarrollo hasta su implementación, monitoreo y análisis. Incluye una variedad de funcionalidades y herramientas para ayudar a las organizaciones a crear, publicar, asegurar, gestionar y analizar APIs de manera efectiva. Esto puede incluir capacidades como el diseño y la documentación de APIs, la gestión de versiones, la seguridad y el control de acceso, la monitorización del rendimiento y el análisis de uso.
- API Gateway: Un API Gateway es un componente específico dentro de una solución de API Management. Funciona como un punto de entrada único para todas las solicitudes de API, proporcionando funciones de enrutamiento, transformación, seguridad, autenticación y autorización. El API Gateway dirige las solicitudes de los clientes a las API correspondientes detrás de él y puede realizar tareas como la terminación SSL, la transformación de protocolos y la gestión de la seguridad. Esencialmente, actúa como una puerta de enlace entre los clientes que consumen las APIs y los servicios que las proporcionan.
- Service Mesh (Malla de Servicios): Se enfoca en la comunicación interna entre servicios dentro de un entorno de microservicios, especialmente en plataformas de contenedores como Kubernetes. Proporciona una forma de controlar y gestionar la comunicación de bajo nivel entre servicios, incluyendo capacidades de descubrimiento de servicios, balanceo de carga, terminación TLS, observabilidad, trazas, y políticas de seguridad y resiliencia como reintentos y cortocircuitos.
- Contenedor: Entorno virtualizado y aislado que permite ejecutar y distribuir aplicaciones de manera consistente en diferentes sistemas operativos y entornos de infraestructura. Los contenedores encapsulan no solo el código de la aplicación, sino también todas las

dependencias y configuraciones necesarias para su funcionamiento. Esto facilita la portabilidad, la escalabilidad y la gestión de las aplicaciones en entornos ágiles, ya que los contenedores pueden desplegarse rápidamente y de manera reproducible en cualquier entorno compatible con la tecnología de contenedores, como Docker o Kubernetes.

- Kubernetes: Es una plataforma de código abierto diseñada para automatizar, desplegar, escalar y operar aplicaciones en contenedores. Desarrollado originalmente por Google y ahora mantenido por la Cloud Native Computing Foundation (CNCF), Kubernetes proporciona un entorno de gestión de contenedores altamente escalable y flexible.
- image-based integration deployment: es una estrategia de despliegue. Se utiliza en entornos modernos de integración y desarrollo, especialmente en arquitecturas contenedorizadas como Docker o Kubernetes. Esta estrategía aprovecha imágenes inmutables para empaquetar el tiempo de ejecución y los artefactos necesarios para la integración, lo que permite despliegues consistentes y escalables. No se considera un patrón en sí mismo, sino más bien un enfoque o método práctico dentro del proceso de integración y despliegue continuo (CI/CD).
- DevOps: es una metodología de desarrollo de software que combina desarrollo (Dev) y operaciones (Ops) con el objetivo de mejorar la colaboración y productividad mediante la automatización del proceso de entrega de software y la integración continua de los cambios. Su propósito es acelerar el tiempo de comercialización de las aplicaciones, manteniendo un alto nivel de calidad y estabilidad operativa.
- Adaptador: Es un componente que actúa como un puente entre diferentes sistemas, aplicaciones o protocolos de comunicación. Su función principal es facilitar la interoperabilidad entre sistemas heterogéneos al convertir los formatos de datos, protocolos de comunicación o interfaces de aplicación de un sistema en los requeridos por otro sistema.
- Arquitectura de Software: “La arquitectura de software es el diseño de más alto nivel de la estructura de un sistema, el cual consiste en un conjunto de patrones y abstracciones que proporcionan un marco claro para la implementación del sistema.” [1]
- Serverless: también conocido como "computación sin servidor", es un modelo de computación en la nube en el que los proveedores de servicios en la nube son responsables de la infraestructura subyacente y la administración de servidores. En el modelo serverless, los desarrolladores pueden centrarse únicamente en escribir y desplegar código, sin preocuparse por la gestión de servidores o la infraestructura subyacente.
- Patrón Arquitectonico: “Los patrones de una arquitectura expresan una organización estructural fundamental o esquema para sistemas complejos. Proporciona un conjunto de subsistemas redefinidos, especifica sus responsabilidades únicas e incluye las reglas y pautas que permiten la toma de decisiones para organizar las relaciones entre ellos. El patrón de arquitectura para un sistema de software ilustra la estructura de nivel macro para toda la solución de software. Un patrón arquitectónico es un conjunto de principios y un patrón de grano grueso que proporciona un marco abstracto para una familia de sistemas. Un patrón arquitectónico mejora la partición y promueve la reutilización del diseño al proporcionar soluciones a problemas recurrentes con frecuencia. Precisamente hablando, un patrón arquitectónico comprende un conjunto de principios que dan forma a una aplicación.”” [3]
- Patrón de Diseño: “Es la solución a un problema de diseño, el cual debe haber comprobado su efectividad resolviendo problemas similares en el pasado, también tiene que ser reutilizable, por lo que se deben poder usar para resolver problemas parecidos en contextos diferentes.”
- Patrones Creacionales: “Son patrones de diseño relacionados con la creación o construcción de objetos. Estos patrones intentan controlar la forma en que los objetos son creados, implementando mecanismos que eviten la creación directa de objetos”. [2]

- Patrones Estructurales: “Son patrones que tienen que ver con la forma en que las clases se relacionan con otras clases. Estos patrones ayudan a dar un mayor orden a nuestras clases ayudando a crear componentes más flexibles y extensibles”.[2]
- Patrones de Comportamiento: “Son patrones que están relacionados con procedimientos y con la asignación de responsabilidad a los objetos. Los patrones de comportamiento engloban también patrones de comunicación entre ellos”. [2]
- Estilo Arquitectónico: “Un estilo arquitectónico es una colección con nombre de decisiones de diseño arquitectónico que son aplicables en un contexto de desarrollo dado, restringen decisiones de diseño arquitectónico que son específicas de un sistema particular dentro de ese contexto, y obtienen cualidades beneficiosas en cada uno sistema resultante.“ [4]
- Arquitectura de Microservicios: “una manera de diseñar aplicaciones de software en servicios independientes” [5]
- Microservicio: "un repositorio de código pequeño, autónomo y desplegado independientemente de los demás". Se desarrollan y diseñan en base a una capacidad de negocio concreta y pueden ser vistos como una aplicación independiente del resto de microservicios, aún cuando coexistan para la implementación del sistema completo. [6]
- Requerimientos funcionales: Un requerimiento funcional representa una característica que debe tener el sistema, la cual es por lo general solicitada de forma explícita. [2]
- Requerimientos no funcionales: Los requerimientos no funcionales son aquellos que no se refieren directamente a una funcionalidad específica del sistema, si no a las propiedades del sistema, como la seguridad, performance, usabilidad, etc. [2]
- Atributos de calidad: Son todos aquellos requerimientos funcionales o no funcionales que tiene un impacto directo sobre la arquitectura del sistema (requisitos arquitectónicamente significativos) y que requieren la atención de un arquitecto. [2]
- Cache: El caché es un componente de hardware o software que almacena datos para que las solicitudes futuras de esos datos se puedan atender con mayor rapidez; los datos almacenados en un caché pueden ser el resultado de un cálculo anterior o el duplicado de datos almacenados en otro lugar, generalmente, da velocidad de acceso más rápido.
- Deuda técnica: Con deuda técnica, se definen los errores, carencias y deficiencias deliberadas o involuntarias en el código que se generan por falta de comunicación, dirección de equipo, cualificación o publicación apresurada de productos y que aumentan constantemente debido a la falta de refactorización.
- Gold Hammer Anti-pattern: El anti patrón Gold Hammer nos habla del uso excesivo de la misma herramienta para resolver cualquier tipo de problema, incluso si hay otra que lo resuelve de mejor manera. Se suele utilizar la siguiente frase para describirlo “Si todo lo que tienes es un martillo, todo parece un clavo”.[2]

5.2. Acrónimos

- API: La interfaz de programación de aplicaciones, conocida también por la sigla API, en inglés, application programming interface. [9]
- DBMS: DataBase Management System
- JWT: JSON Web Token (abreviado JWT) es un estándar abierto basado en JSON propuesto por IETF (RFC 7519) para la creación de tokens de acceso que permiten la propagación de identidad y privilegios o claims en inglés. Por ejemplo, un servidor podría generar un token indicando que el usuario tiene privilegios de administrador y proporcionar a un cliente. El cliente entonces podría utilizar el token para probar que está actuando como un administrador en el cliente o en otro sistema. El token está firmado por la clave del servidor, así que el cliente y el servidor son ambos capaces de verificar que el token es legítimo. Los JSON Web Tokens están diseñados para ser

compactos, poder ser enviados en las URLs -URL-safe- y ser utilizados en escenarios de Single Sign-On (SSO). Los privilegios de los JSON Web Tokens pueden ser utilizados para propagar la identidad de usuarios como parte del proceso de autenticación entre un proveedor de identidad y un proveedor de servicio, o cualquiera otro tipo de privilegios requeridos por procesos empresariales. [10]

6. Descripción del Contexto

La plataforma Catastro Virtual surge como una iniciativa estratégica de la Alcaldía de Medellín, liderada por la Secretaría de Gestión y Control Territorial, con el objetivo de modernizar los procesos catastrales, facilitar el acceso a los servicios públicos y fortalecer la interacción entre la ciudadanía y la administración Distrital. Esta iniciativa se enmarca dentro del programa de fortalecimiento del Catastro Distrital, alineado con las políticas de gobierno digital y el Plan de Desarrollo del Distrito de Ciencia, Tecnología e Innovación.

6.1. Situación Actual

En la actualidad, los trámites relacionados con el catastro presentan desafíos significativos, tales como:

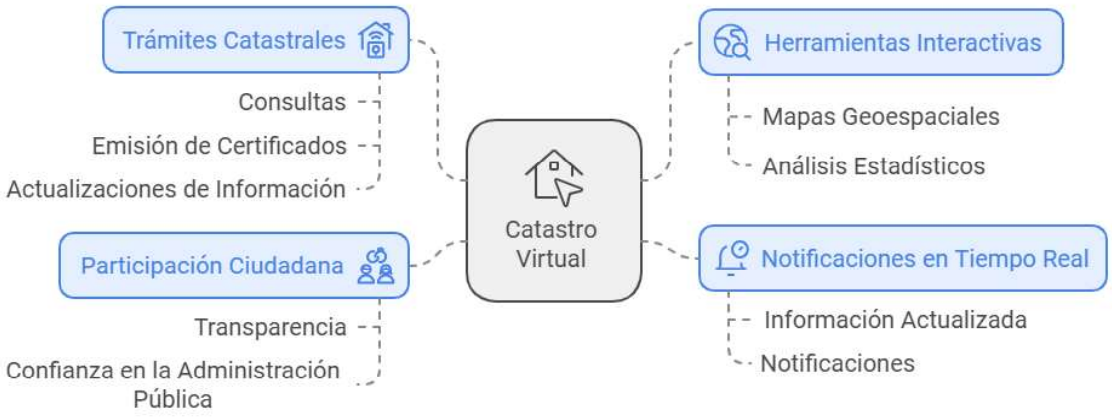
- Procesos manuales que implican altos costos operativos, tiempos de respuesta prolongados y errores administrativos que derivan en salidas no conformes en los procesos, generando riesgo de imagen.
- Dificultades de acceso a los servicios catastrales, especialmente para ciudadanos en zonas periféricas o con limitaciones de movilidad.
- Baja transparencia en el seguimiento de trámites y la disponibilidad de información en tiempo real.
- Necesidad de mejorar la protección de los datos personales de los ciudadanos.

Estos desafíos limitan la capacidad de la administración Distrital para brindar servicios eficientes, accesibles y seguros, y dificultan la participación ciudadana en la gestión territorial.

6.2. Visión del Proyecto

El Catastro Virtual busca resolver estas problemáticas mediante la implementación de una única plataforma digital que incorpore automatización, inteligencia artificial con altos estándares de seguridad, accesible desde dispositivos móviles y portales web, que permita a los ciudadanos:

- Realizar trámites catastrales como consultas, emisión de certificados con tecnologías de seguridad como BlockChain y firmas digitales; radicación de solicitudes y actualizaciones de información de propiedad de manera remota.
- Acceder a herramientas interactivas como mapas geoespaciales y análisis estadísticos.
- Obtener información actualizada y notificaciones en tiempo real sobre sus solicitudes.
- Participar activamente en la gestión del territorio, fortaleciendo la transparencia y la confianza en la administración pública.



6.3. Ecosistema de la Plataforma

El desarrollo de esta plataforma se inserta dentro de un ecosistema tecnológico más amplio que integra:

- Interoperabilidad con otras dependencias municipales: Facilitando el acceso a datos relevantes y asegurando la consistencia de la información.
- Cumplimiento normativo: Asegurando la protección de datos y la transparencia en el manejo de la información.
- Adopción de tecnologías modernas: Uso de herramientas avanzadas como microservicios, contenedores, y sistemas de monitoreo para garantizar una solución escalable y resiliente.

6.4. Impacto Esperado

El Catastro Virtual permitirá a la Alcaldía de Medellín:

- Reducir los costos operativos mediante la automatización de procesos, mejorando la experiencia del usuario.

- Ampliar la cobertura de los servicios catastrales, alcanzando a más ciudadanos de manera eficiente.
- Incrementar la confianza de la ciudadanía en la gestión pública gracias a la transparencia y seguridad de la plataforma, además de la alineación con las normativas vigentes, reduciendo riesgos legales y fortaleciendo la confianza en la gestión pública
- Posicionar a Medellín como un referente en innovación tecnológica y modernización de servicios públicos en Colombia.
- Impacto ambiental, validar cantidad de tramites que hacen en papel

Esta plataforma no solo modernizará la gestión catastral, sino que también se convertirá en un ejemplo de cómo las tecnologías digitales pueden transformar la interacción entre la administración pública y la ciudadanía, fomentando una ciudad más inclusiva, transparente y eficiente.

7. Requisitos Funcionales y No Funcionales

Los requerimientos funcionales y no funcionales son elementos clave en el diseño y desarrollo de cualquier sistema tecnológico, ya que definen las capacidades y restricciones que debe cumplir la solución. Los requerimientos funcionales describen qué debe hacer el sistema desde el punto de vista del usuario o de los procesos que soportan alineadas con las disposiciones establecidas. En el caso de la plataforma Catastro Virtual, incluyen funcionalidades como el registro de usuarios, la consulta y emisión de certificados catastrales, la gestión de mutaciones catastrales en sus diferentes clases (primera a quinta), el seguimiento de trámites, notificación en tiempo real entre otros. Por otro lado, los requerimientos no funcionales establecen las condiciones de calidad y las características técnicas que debe cumplir el sistema, como la seguridad, el rendimiento, la escalabilidad, la usabilidad y la disponibilidad. Estos requerimientos garantizan que la plataforma no solo cumpla con sus objetivos funcionales, sino que lo haga de manera eficiente, segura y confiable para los usuarios finales y las partes interesadas.

7.1. Requerimientos no funcionales:

A continuación, se presenta un listado de los requerimientos no funcionales clave para la plataforma Catastro Virtual:

1. Seguridad:
    - Garantizar la confidencialidad, integridad y disponibilidad de los datos de los ciudadanos.
    - Implementar autenticación y autorización robustas mediante protocolos estándar como OAuth2 y OpenID Connect.
    - Proteger los datos sensibles mediante cifrado en tránsito (TLS) y en reposo.
2. Escalabilidad:
    - Permitir la ampliación de la capacidad del sistema para manejar un mayor número de usuarios concurrentes y transacciones sin degradar el rendimiento.
3. Rendimiento:
    - Responder a las solicitudes del usuario en un tiempo promedio inferior a 2 segundos bajo carga normal.
    - Mantener un tiempo de respuesta inferior a 5 segundos incluso bajo picos de carga.
4. Alta Disponibilidad:
    - Garantizar una disponibilidad del sistema del 99.9% (máximo 8 horas de inactividad al año).
    - Implementar mecanismos de failover y redundancia para minimizar interrupciones.
5. Compatibilidad:
    - Asegurar que la plataforma sea accesible desde los principales navegadores web (Chrome, Edge, Safari y Firefox) y dispositivos móviles con sistemas operativos iOS y Android.
6. Usabilidad:
    - Diseñar interfaces intuitivas y accesibles para usuarios con diferentes niveles de conocimiento tecnológico.
    - Cumplir con los estándares de accesibilidad (WCAG 2.1).

7. Monitoreo y Registro:
    - Proveer capacidades de monitoreo en tiempo real para métricas clave como el tiempo de respuesta, la carga del servidor y el uso de recursos.
    - Implementar registros centralizados de auditoría para todas las transacciones y eventos críticos del sistema.
8. Mantenibilidad:
    - Diseñar el sistema de manera modular para facilitar actualizaciones, correcciones y la incorporación de nuevas funcionalidades.
    - Documentar el código y las configuraciones de manera clara y estandarizada.
9. Portabilidad:
    - Garantizar que los componentes del sistema puedan desplegarse en múltiples entornos de infraestructura (nube pública, privada o híbrida).
    - Utilizar contenedores para simplificar la portabilidad del sistema.
10. Interoperabilidad:
    - Asegurar la integración con otros sistemas municipales e institucionales mediante APIs estandarizadas (REST o GraphQL).
    - Facilitar el intercambio de datos con plataformas externas siguiendo estándares abiertos.
11. Confiabilidad:
    - Prevenir pérdida de datos mediante estrategias de respaldo automático y recuperación ante desastres.
    - Asegurar la consistencia de los datos en todas las transacciones.
12. Cumplimiento Normativo:
    - Adherirse a las normativas locales e internacionales de protección de datos, como la Ley 1581 de 2012 en Colombia y el GDPR (Reglamento General de Protección de Datos);

    - Decreto 148 de 2020: Regula la implementación y operación de sistemas catastrales en Colombia, promoviendo la interoperabilidad y la articulación institucional para garantizar la calidad y accesibilidad de la información.

    - Resolución Conjunta IGAC No. 1101 SNR No. 11344 de 2020: Establece lineamientos técnicos y operativos para la gestión de datos catastrales y registrales en el marco del Catastro Multipropósito.

    - Resolución 1438-2017: Define estándares técnicos para la generación, actualización y administración de la información catastral en Colombia.

    - Resolución 1149 de 2021: Regula los procedimientos para la administración de información predial en el Catastro priorizando la precisión y la calidad de los datos.

    - Ley 1450 de 2011: Plan Nacional de Desarrollo que incluye disposiciones relacionadas con la modernización de los sistemas catastrales y registrales en Colombia.

    - Ley 1437 de 2011: Código de Procedimiento Administrativo y de lo Contencioso Administrativo, que establece principios y normas aplicables a los procesos administrativos y la gestión de información pública.

    - LADM_COL Versión 4.1 -Resolución Conjunta IGAC 1456 SNR 09844 de 12 de septiembre del 2024)

7.2. Patrones y buenas prácticas

Los patrones y buenas prácticas desempeñan un papel fundamental en el diseño e implementación de sistemas tecnológicos robustos, escalables y sostenibles. Para la plataforma Catastro Virtual, se adoptarán patrones de diseño como Microservicios, que promueven la modularidad y la independencia de componentes, y Circuit Breaker, para garantizar la resiliencia y la tolerancia a fallos en los servicios. Además, se seguirán prácticas recomendadas como el principio de Segregación de Responsabilidades, asegurando que cada componente cumpla con una función específica, y el uso de Infraestructura como Código (IaC) para la automatización del despliegue y configuración en entornos consistentes. Estas prácticas, complementadas con pruebas automatizadas, monitoreo continuo y estrategias de integración continua y entrega continua (CI/CD), asegurarán que la plataforma no solo cumpla con los requisitos funcionales y no funcionales, sino que también ofrezca una experiencia confiable, segura y eficiente a los usuarios.

7.3. The Twelve-Factor App

La plataforma Catastro Virtual adoptará los principios establecidos en el marco de The Twelve-Factor App, un conjunto de buenas prácticas diseñadas para desarrollar aplicaciones modernas, escalables y portables en entornos de nube. Estos principios garantizan que la plataforma sea fácil de construir, desplegar y mantener. Algunos factores clave incluyen la externalización de configuraciones para asegurar la separación entre el código y los entornos (por ejemplo, gestión de credenciales con herramientas como HashiCorp Vault), la implementación de procesos de construcción, liberación y ejecución claramente definidos para habilitar un pipeline CI/CD

eficiente, y el diseño de aplicaciones como procesos stateless (sin estado) para mejorar la escalabilidad y el manejo de cargas dinámicas. Además, el uso de logs como flujos de eventos y la dependencia explícita de servicios externos a través de gestión de dependencias (como bibliotecas y API Gateway) asegurarán una operación resiliente y monitoreada en tiempo real. Estos patrones alineados con Twelve-Factor App permiten construir una plataforma que aprovecha al máximo las capacidades de tecnologías como Kubernetes y Docker, garantizando su robustez y sostenibilidad a largo plazo.

Los doce factores:

1. Codebase: Utiliza un repositorio de código único y versionado para cada aplicación.
2. Dependencies: Declara y aísla explícitamente las dependencias de la aplicación.
3. Config: Almacena la configuración en el entorno, evitando el almacenamiento de configuraciones en el código.
4. Backing services: Trata los servicios de respaldo como recursos adjuntos a la aplicación que pueden ser cambiados fácilmente.
5. Build, release, run: Separación de los procesos de construcción, liberación y ejecución.
6. Processes: Ejecuta la aplicación como uno o más procesos que son efímeros, independientes y desechables.
7. Port binding: Exporta servicios a través de enlaces de puerto, permitiendo la comunicación con otros procesos.
8. Concurrency: Escala horizontalmente a través del modelo de proceso.
9. Disposability: Maximiza la robustez con inicio rápido y apagado gracioso.
10. Dev/prod parity: Mantiene la paridad entre los entornos de desarrollo, prueba y producción.
11. Logs: Trata los logs como flujos de eventos, agregándolos y almacenándolos en un sistema de gestión de logs.
12. Admin processes: Ejecuta tareas de administración / gestión como procesos únicos, independientes de la aplicación.

Beneficios:

- Portabilidad: Las aplicaciones desarrolladas siguiendo los principios de "The Twelve-Factor App" son altamente portátiles y pueden ser ejecutadas en una variedad de entornos de ejecución.
- Escalabilidad: La metodología fomenta la construcción de aplicaciones que pueden escalar fácilmente para manejar aumentos en la carga de trabajo.
- Mantenibilidad: La separación de preocupaciones y la estandarización de prácticas de desarrollo facilitan la mantenibilidad y la colaboración entre equipos de desarrollo.
- Confiabilidad: La metodología promueve prácticas como la gestión de configuraciones externas y el tratamiento de servicios de respaldo como recursos adjuntos, lo que contribuye a la fiabilidad de las aplicaciones.

Ejemplo:

Imagina una aplicación web desarrollada siguiendo los principios de "The Twelve-Factor App". La aplicación se despliega en un entorno de contenedorización, donde cada uno de los servicios se ejecuta como un contenedor Docker independiente. La configuración de la aplicación se administra a través de variables de entorno, lo que permite una fácil configuración en diferentes entornos de ejecución. Los logs de la aplicación se envían a un servicio centralizado de gestión de logs (Log-based monitoring), donde pueden ser monitoreados y analizados para identificar problemas y realizar mejoras en la aplicación. Este enfoque asegura que la aplicación sea portátil, escalable y fácil de mantener en entornos de nube.

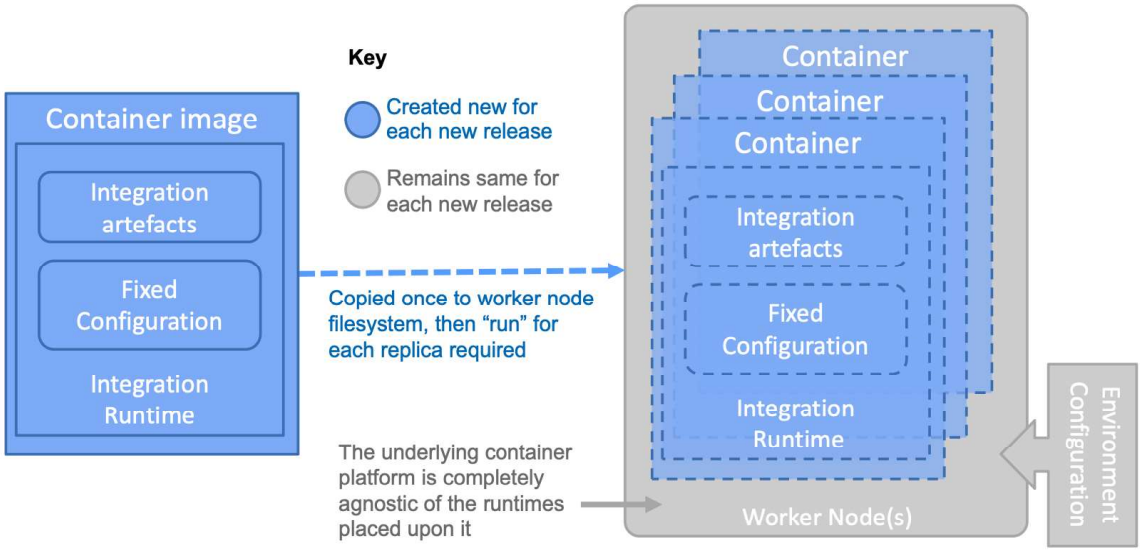
8. Estrategias de despliegue

8.1. Image-based integration deployment

El despliegue basado en imágenes es un mecanismo utilizado para el empaquetado de todas las dependencias locales en una imagen inmutable. Este enfoque permite que la imagen contenga los artefactos de integración, el tiempo de ejecución del producto de integración y lo suficiente del sistema operativo para funcionar de manera independiente en un entorno contenedorizado.

Esta metodología asegura que todo el paquete de integración sea consistente y se despliegue rápidamente, ofreciendo varios beneficios para los entornos contenedorizados. Por ejemplo, estos entornos pueden desplegar y administrar integraciones sin necesidad de conocer detalles específicos sobre el producto de integración, lo que mejora la confianza en las pruebas, facilita la recreación de ambientes para pruebas funcionales y de rendimiento, y simplifica el diagnóstico de problemas.

Además, a diferencia del despliegue tradicional donde los servidores de integración requieren administración continua y directa cuando están activos, el despliegue basado en imágenes permite un enfoque más modular y menos intrusivo. Una vez que se instala el producto y se añaden los artefactos de la aplicación capturando todo en una imagen de contenedor, esta imagen se convierte en una "caja negra" que no requiere conocimientos específicos del tiempo de ejecución o las integraciones que contiene para su funcionamiento .



8.2. Monitoreo basado en logs (Log-based monitoring)

El monitoreo basado en logs es una técnica esencial en el contexto de las integraciones nativas de la nube o integraciones ágiles basadas en microservicios. Consiste en recopilar, analizar y

extraer información relevante de los registros generados por las aplicaciones y servicios desplegados en entornos de nube u OnPremise. Estos registros contienen valiosa información sobre el comportamiento, el rendimiento y el estado de la aplicación, lo que permite a los equipos de operaciones y desarrollo detectar y solucionar problemas, así como optimizar el rendimiento del sistema de manera proactiva.

Flujo de trabajo típico:

1. Generación de Logs: Las aplicaciones y servicios generan registros que registran eventos, acciones y métricas relevantes durante su funcionamiento. Estos registros pueden contener información sobre errores, excepciones, transacciones, solicitudes HTTP, tiempos de respuesta, entre otros datos útiles.

2. Recolección de Logs: Los registros generados por las aplicaciones se recopilan y almacenan centralmente en un repositorio de logs, como Prometheus, AWS Cloud Watch Elasticsearch, Splunk, Loggly, o en sistemas de almacenamiento de objetos como Amazon S3, MinIO.

3. Análisis de Logs: Los registros recopilados se analizan utilizando herramientas de análisis de logs para identificar patrones, tendencias y anomalías. Esto puede incluir la búsqueda de palabras clave, la correlación de eventos, la creación de métricas y la generación de informes.

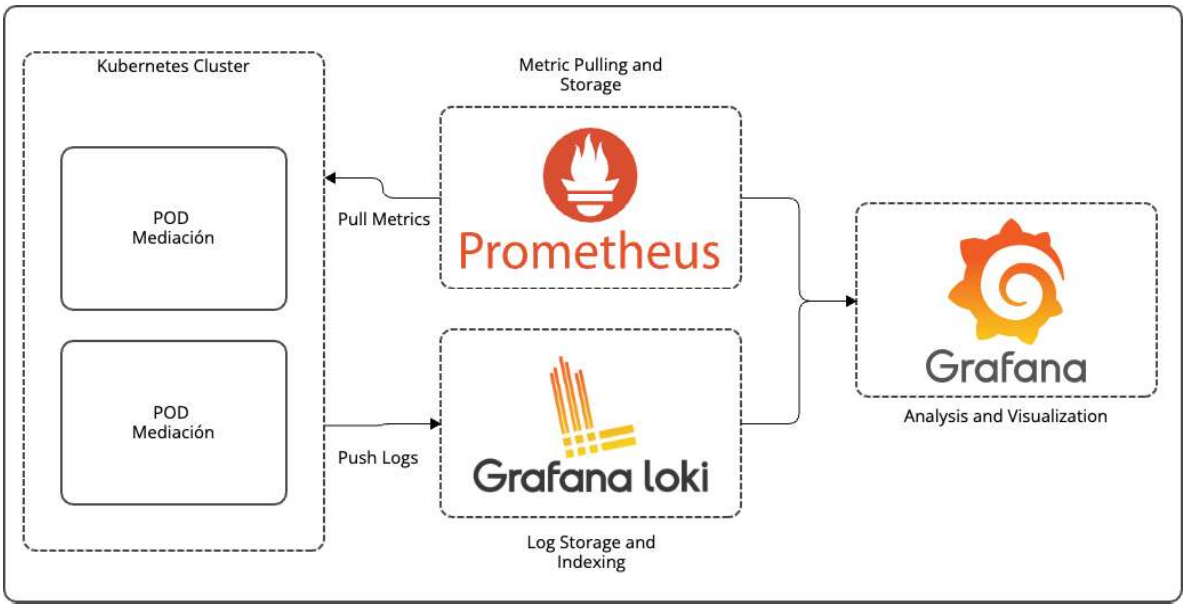
4. Alertas y Notificaciones: Se configuran alertas y notificaciones basadas en los resultados del análisis de logs para informar al equipo de operaciones sobre eventos importantes, como errores críticos, tiempos de respuesta lentos o caídas del sistema.

5. Resolución de Problemas: Los equipos de operaciones y desarrollo utilizan la información extraída de los logs para diagnosticar y solucionar problemas de manera rápida y eficiente. Esto puede implicar la identificación de la causa raíz de un problema, la implementación de correcciones y la validación de la efectividad de las soluciones.

6. Optimización del Rendimiento: El análisis continuo de los logs permite identificar áreas de mejora en el rendimiento y la eficiencia de la aplicación. Esto puede incluir la optimización de consultas de bases de datos, la reducción de cuellos de botella y la implementación de mejoras en la infraestructura.

Ejemplo:

En una aplicación web desplegada en la nube, el monitoreo basado en logs puede ayudar a detectar y resolver problemas de rendimiento. Si los registros muestran un aumento inusual en los tiempos de respuesta de las solicitudes HTTP, el equipo de operaciones puede recibir una alerta y comenzar a investigar la causa subyacente. Al analizar los logs, descubren que un servicio de base de datos está experimentando un aumento en la carga debido a consultas mal optimizadas. Con esta información, el equipo puede tomar medidas correctivas, como ajustar las consultas, escalar horizontalmente la base de datos o implementar una memoria caché, para mejorar el rendimiento y garantizar una experiencia de usuario óptima. Ver Figura 2.



9. API

9.1. Request/Response

El patrón de API Request/Response es uno de los patrones más comunes y fundamentales en el diseño de APIs. En este patrón, un cliente envía una solicitud a un servidor a través de una API y espera recibir una respuesta del servidor en consecuencia. A continuación se encuentra una descripción detallada del patrón:

Descripción:

1. Cliente: El cliente es la entidad que inicia la comunicación enviando una solicitud al servidor a través de la API. Puede ser cualquier aplicación o sistema que desee acceder a los servicios proporcionados por el servidor.

2. Solicitud (Request): La solicitud es un mensaje enviado por el cliente al servidor que especifica la acción deseada, así como cualquier dato adicional necesario para completar la acción. La solicitud puede contener información como parámetros de consulta, datos de formulario, encabezados de solicitud, etc.

3. Servidor: El servidor es la entidad que recibe las solicitudes del cliente, procesa la solicitud y genera una respuesta en consecuencia. Puede ser cualquier sistema o aplicación que proporcione servicios a través de una API.

4. Respuesta (Response): La respuesta es un mensaje enviado por el servidor al cliente en respuesta a una solicitud recibida. La respuesta contiene el resultado de la acción solicitada, así como cualquier otro dato relevante, como códigos de estado, encabezados de respuesta, y el cuerpo de la respuesta que puede contener datos estructurados.

5. Protocolo de Comunicación: El patrón Request/Response puede implementarse sobre varios protocolos de comunicación, como HTTP, WebSocket, gRPC, entre otros. El

protocolo especifica las reglas y convenciones para la comunicación entre el cliente y el servidor, incluyendo cómo se estructuran las solicitudes y respuestas, y cómo se manejan los errores y las excepciones.

Flujo de trabajo típico:

1. El cliente envía una solicitud al servidor a través de la API, especificando la acción deseada y cualquier dato adicional necesario.
2. El servidor recibe la solicitud y la procesa, realizando las operaciones correspondientes según la acción solicitada.
3. Una vez completada la operación, el servidor genera una respuesta que incluye el resultado de la acción, así como cualquier otro dato relevante.
4. El servidor envía la respuesta de vuelta al cliente a través de la API.
5. El cliente recibe la respuesta y la procesa según sea necesario para continuar con su flujo de trabajo.

Ejemplo:

Supongamos que tienes una API de gestión de usuarios y deseas recuperar los detalles de un usuario específico. El flujo de trabajo utilizando el patrón Request/Response sería el siguiente:

1. El cliente envía una solicitud GET al servidor a través de la API, especificando la URL del recurso del usuario deseado, por ejemplo, `/users/123`.
2. El servidor recibe la solicitud, recupera los detalles del usuario con el identificador 123 de la base de datos.
3. Una vez recuperados los detalles del usuario, el servidor genera una respuesta con los datos del usuario y un código de estado 200 (OK).
4. El servidor envía la respuesta de vuelta al cliente.
5. El cliente recibe la respuesta y puede procesar los datos del usuario, como mostrarlos en la interfaz de usuario de la aplicación.

Este es solo un ejemplo básico del patrón Request/Response en acción. Este patrón es ampliamente utilizado en el diseño de APIs para proporcionar una comunicación simple y eficaz entre clientes y servidores.

9.2. Request/acknowledge/poll

El patrón de API Request/Acknowledge/Poll es un enfoque comúnmente utilizado para la comunicación entre clientes y servidores en sistemas distribuidos. Este patrón se emplea cuando la sincronización de los eventos o el estado entre el cliente y el servidor es crucial, y el servidor puede no ser capaz de proporcionar una respuesta inmediata a una solicitud.

Descripción:

1. Solicitud (Request):
   - El cliente envía una solicitud al servidor a través de la API, solicitando una acción o un recurso específico.
   - La solicitud puede contener información adicional sobre la acción requerida o los datos necesarios para completarla.

2. Acknowledgement (Acknowledge):

- Una vez que el servidor recibe la solicitud, envía un acuse de recibo al cliente para confirmar que ha recibido la solicitud correctamente.
- El acuse de recibo puede incluir un identificador único o un token que el cliente puede utilizar para identificar y rastrear la solicitud posteriormente.

3. Polling (Poll):
   - Después de recibir el acuse de recibo, el cliente inicia un proceso de "sondeo" o "consulta" periódica al servidor para verificar el estado o el resultado de la solicitud original.
   - El cliente envía solicitudes de sondeo repetidas al servidor a intervalos regulares hasta que recibe una respuesta definitiva o una notificación de que la solicitud ha sido completada.

Flujo de trabajo típico:

1. El cliente envía una solicitud al servidor a través de la API, solicitando una acción específica y proporcionando los datos necesarios.
2. El servidor recibe la solicitud y envía un acuse de recibo al cliente para confirmar que la solicitud ha sido recibida correctamente.
3. El cliente inicia un proceso de sondeo periódico al servidor para verificar el estado o el resultado de la solicitud original.
4. El servidor procesa la solicitud en segundo plano y actualiza su estado o resultado según sea necesario.
5. El servidor responde a las solicitudes de sondeo del cliente con información actualizada sobre el estado o el resultado de la solicitud original.
6. El cliente continúa sondeando al servidor hasta que recibe una respuesta definitiva que indica que la solicitud ha sido completada o que se ha producido un error.

Ejemplo:

Supongamos que un cliente realiza una solicitud para iniciar un proceso de generación de informes largos a través de una API. El servidor recibe la solicitud y envía un acuse de recibo al cliente. Luego, el cliente inicia un proceso de sondeo periódico para verificar el estado del informe. Mientras tanto, el servidor procesa el informe en segundo plano y actualiza su estado. Cuando el informe está listo, el servidor responde a las solicitudes de sondeo del cliente con la URL del informe generado, y el cliente puede descargar el informe.

Este es solo un ejemplo básico del patrón Request/Acknowledge/Poll en acción. Este patrón es útil en situaciones donde la respuesta inmediata del servidor no es posible o cuando se requiere una sincronización continua entre el cliente y el servidor para verificar el estado o el resultado de una operación.

9.3. Request/acknowledge/callback

El patrón de API Request/Acknowledge/Callback es un enfoque utilizado en sistemas distribuidos para manejar la asincronía en la comunicación entre clientes y servidores. Este patrón se emplea cuando el servidor necesita tiempo para procesar una solicitud y el cliente no puede esperar una respuesta inmediata. Aquí tienes una descripción detallada del patrón:

Descripción:

1. Solicitud (Request):

- El cliente envía una solicitud al servidor a través de la API, solicitando una acción o un servicio específico.
- La solicitud puede contener datos adicionales necesarios para completar la acción solicitada.

2. Acknowledgement (Acknowledge):
   - Una vez que el servidor recibe la solicitud, envía un acuse de recibo al cliente para confirmar que ha recibido la solicitud correctamente.
   - El acuse de recibo puede incluir un identificador único o un token que el cliente puede utilizar para identificar y rastrear la solicitud posteriormente.

3. Callback (Callback):
   - Después de recibir la solicitud y enviar el acuse de recibo, el servidor procesa la solicitud en segundo plano.
   - Una vez que se completa el procesamiento de la solicitud, el servidor llama de vuelta al cliente utilizando una dirección (URL) predefinida proporcionada por el cliente en la solicitud original.
   - El servidor envía los resultados de la solicitud al cliente a través de la llamada de retorno, que puede contener información sobre el estado de la solicitud, los datos procesados o cualquier otro resultado relevante.

Flujo de trabajo típico:

1. El cliente envía una solicitud al servidor a través de la API, solicitando una acción específica y proporcionando los datos necesarios.
2. El servidor recibe la solicitud y envía un acuse de recibo al cliente para confirmar que la solicitud ha sido recibida correctamente.
3. El servidor procesa la solicitud en segundo plano mientras el cliente continúa con otras operaciones.
4. Una vez completado el procesamiento de la solicitud, el servidor llama de vuelta al cliente utilizando la dirección (URL) proporcionada por el cliente en la solicitud original.
5. El servidor envía los resultados de la solicitud al cliente a través de la llamada de retorno, que puede incluir información sobre el estado de la solicitud o cualquier otro resultado relevante.
6. El cliente recibe la llamada de retorno del servidor y puede procesar los resultados de la solicitud según sea necesario.

Ejemplo:

Supongamos que un cliente realiza una solicitud para procesar una gran cantidad de datos a través de una API. El servidor recibe la solicitud, envía un acuse de recibo al cliente y comienza a procesar los datos en segundo plano. Una vez que se completa el procesamiento, el servidor llama de vuelta al cliente utilizando una dirección proporcionada por el cliente en la solicitud original. El servidor envía los resultados del procesamiento de datos al cliente a través de la llamada de retorno, permitiendo que el cliente continúe con su flujo de trabajo basado en los resultados obtenidos.

Este es solo un ejemplo básico del patrón Request/Acknowledge/Callback en acción. Este patrón es útil en situaciones donde la respuesta inmediata del servidor no es posible y se necesita una comunicación asincrónica entre el cliente y el servidor para manejar las operaciones de larga duración.

9.4. Consumer Adapter

El patrón de API Consumer Adapter es una técnica utilizada en el diseño de APIs para permitir la interoperabilidad entre sistemas heterogéneos. Este patrón se utiliza cuando se necesita adaptar la interfaz de una API consumida por un cliente para que sea compatible con el formato o protocolo esperado por el cliente. Aquí tienes una descripción detallada del patrón:

Descripción:

1. API del Proveedor (Provider API):
   - Es la API proporcionada por el proveedor del servicio o sistema al que el cliente desea acceder.
   - La API del proveedor puede estar diseñada utilizando un formato de datos o protocolo específico que puede no ser compatible con el cliente.

2. Consumer Adapter:
   - Es un componente intermediario que se encuentra entre el cliente y la API del proveedor.
   - El Consumer Adapter actúa como un adaptador que traduce las solicitudes y respuestas entre el formato esperado por el cliente y el formato utilizado por la API del proveedor.
   - Este adaptador puede realizar tareas como la conversión de datos, la transformación de mensajes, la autenticación, la autorización y la gestión de errores para garantizar una integración sin problemas entre el cliente y la API del proveedor.

3. Cliente (Client):
   - Es la entidad que consume los servicios proporcionados por la API del proveedor a través del Consumer Adapter.
   - El cliente puede ser cualquier aplicación, sistema o componente que necesite acceder a los servicios proporcionados por el proveedor.

Flujo de trabajo típico:

1. El cliente envía una solicitud al Consumer Adapter utilizando la interfaz de la API diseñada específicamente para el cliente.
2. El Consumer Adapter recibe la solicitud del cliente y la traduce al formato o protocolo esperado por la API del proveedor.
3. El Consumer Adapter envía la solicitud adaptada a la API del proveedor.
4. La API del proveedor procesa la solicitud y envía una respuesta al Consumer Adapter.
5. El Consumer Adapter recibe la respuesta del proveedor y la traduce al formato o protocolo esperado por el cliente.
6. El Consumer Adapter envía la respuesta adaptada al cliente.
7. El cliente recibe la respuesta del Consumer Adapter y puede procesarla según sea necesario.

Ejemplo:

Supongamos que un cliente desea consumir un servicio proporcionado por una API RESTful que utiliza un formato de datos JSON para las solicitudes y respuestas. Sin embargo, el servicio que el cliente desea acceder proporciona una API SOAP que utiliza XML como formato de datos. En este caso, el cliente puede utilizar un Consumer Adapter que traduce las solicitudes y

respuestas entre JSON y XML, permitiendo al cliente consumir el servicio a través de su interfaz de API diseñada específicamente.

Este es solo un ejemplo básico del patrón de API Consumer Adapter en acción. Este patrón es útil cuando se necesita adaptar la interfaz de una API consumida por un cliente para que sea compatible con el formato o protocolo esperado por el cliente, facilitando así la interoperabilidad entre sistemas heterogéneos.

10. File Transfer

10.1. Buenas prácticas para nombrado de archivos

Una buena práctica para nombrar archivos en un patrón de transferencia de archivos es seguir un enfoque que proporcione información clara y significativa sobre el contenido y el propósito del archivo. Aquí hay algunas pautas que se deben seguir:

1. Simplicidad y Consistencia: Mantén los nombres de archivo simples y consistentes en toda tu aplicación. Utiliza convenciones de nomenclatura claras y fáciles de entender que sean consistentes con las convenciones de nomenclatura utilizadas en tu organización.

2. Inclusión de Información Relevante: Incluye información relevante en el nombre del archivo, como el tipo de datos, la fecha de creación o modificación, el origen o destino del archivo, y cualquier otro metadato que sea útil para su identificación y procesamiento.

3. Evitar Caracteres Especiales: Evita el uso de caracteres especiales, espacios en blanco o caracteres no ASCII en los nombres de archivo, ya que pueden causar problemas de compatibilidad y dificultar la manipulación de archivos en algunos sistemas.

4. Longitud Limitada: Limita la longitud de los nombres de archivo para evitar problemas de truncamiento en sistemas que imponen restricciones en la longitud de los nombres de archivo.

5. Utilización de Prefijos o Sufijos: Considera el uso de prefijos o sufijos en los nombres de archivo para proporcionar información adicional, como el tipo de archivo, el estado del archivo (por ejemplo, procesado o no procesado), o la versión del archivo.

6. Versionamiento: Si es necesario manejar múltiples versiones de un archivo, considera incluir información de versión en el nombre del archivo para distinguir entre las diferentes versiones.

7. Utilización de Separadores: Utiliza separadores claros y consistentes para separar diferentes partes del nombre del archivo, como guiones (-), guiones bajos (_), puntos (.) u otros caracteres adecuados.

8. Documentación: Documenta las convenciones de nomenclatura utilizadas en tu aplicación, incluyendo ejemplos y explicaciones claras sobre el significado de cada parte del nombre del archivo, para facilitar su comprensión y mantenimiento por parte de otros desarrolladores.

Siguiendo estas prácticas, puedes crear nombres de archivo significativos y fáciles de entender que faciliten la gestión y el procesamiento de archivos en tu aplicación de transferencia de archivos.

Ejemplos de Nombres de Archivo:

1. customer_data_20220503.csv
   - Nombre de archivo que contiene datos de clientes, con la fecha de creación (2022-05-03) incluida en el nombre.

2. invoices_processed_v2.xlsx
   - Nombre de archivo que representa facturas procesadas, con la versión 2 (v2) indicada en el nombre.

3. inventory_update_20220503.txt
   - Nombre de archivo que contiene actualizaciones de inventario, con la fecha de creación (2022-05-03) incluida en el nombre.

4. sales_report_weekly_2022-05-03.pdf
   - Nombre de archivo que representa un informe de ventas semanal, con la fecha del informe (2022-05-03) incluida en el nombre.

5. order_123456_pending.xml
   - Nombre de archivo que representa una orden de compra pendiente con el número de orden (123456) incluido en el nombre.

6. customer_data_errors_20220503.log
   - Nombre de archivo que contiene registros de errores al procesar datos de clientes, con la fecha de creación (2022-05-03) incluida en el nombre.

7. invoices_failed_processing_v2.xlsx
   - Nombre de archivo que representa facturas que no pudieron ser procesadas, con la versión 2 (v2) indicada en el nombre.

8. inventory_update_errors_20220503.log
   - Nombre de archivo que contiene registros de errores al procesar actualizaciones de inventario, con la fecha de creación (2022-05-03) incluida en el nombre.

Al incluir nombres de archivo para aquellos que no se pudieron procesar, facilitas la identificación y resolución de problemas en tu proyecto de integración. Esto contribuye a una gestión más efectiva y a una mejora en la calidad de los datos y procesos.

10.2. Buenas prácticas para estructura de carpetas

1. Carpeta de Entrada (input):
   - Descripción: Esta carpeta se utiliza para almacenar los archivos que deben ser procesados por el sistema.
   - Ubicación: `/input` (o cualquier ubicación definida según la configuración del sistema).

- Contenido Esperado: Archivos de datos en diversos formatos (por ejemplo, CSV, XML, JSON, Excel, etc.) que contienen la información que debe ser procesada.

2. Carpeta de Salida (output):
   - Descripción: Esta carpeta se utiliza para almacenar los archivos procesados exitosamente.
   - Ubicación: `/output` (o cualquier ubicación definida según la configuración del sistema).
   - Contenido Esperado: Archivos procesados y transformados que representan el resultado de los procesos de integración.

3. Carpeta de Errores (error):
   - Descripción: Esta carpeta se utiliza para almacenar los archivos que no pudieron ser procesados debido a algún error.
   - Ubicación: `/error` (o cualquier ubicación definida según la configuración del sistema).
   - Contenido Esperado: Archivos que contienen registros de errores, mensajes de registro o información de depuración relacionada con los archivos que no pudieron ser procesados.

Convenciones de Nomenclatura:

1. Archivos de Entrada:
   - Utilizar nombres de archivos descriptivos y significativos que reflejen el contenido y el propósito del archivo.
   - Mantener la consistencia en la convención de nomenclatura utilizada para los archivos de entrada.

2. Archivos de Salida:
   - Utilizar nombres de archivos que indiquen claramente el resultado del proceso de procesamiento o transformación.
   - Incluir sufijos o prefijos relevantes para distinguir los archivos de salida de los archivos de entrada.

3. Archivos de Errores:
   - Utilizar nombres de archivos que indiquen claramente que contienen registros de errores o información de depuración.
   - Incluir la fecha o el timestamp en el nombre del archivo para facilitar la identificación y el seguimiento de los errores.

Ejemplo de Estructura de Carpetas:

- integration_project/
  - input/
    - customer_data_20240514.csv
    - sales_orders_20240514.xml
  - output/
    - processed_customer_data_20240514.xlsx
    - processed_sales_orders_20240514.json
  - error/
    - customer_data_errors_20220503.log
    - sales_orders_errors_20220503.log

Al seguir este estándar de manejo de carpetas, facilitas la organización, identificación y gestión de archivos en tu proyecto de integración. Esto contribuye a una operación más efectiva y eficiente del sistema, permitiendo una fácil identificación y resolución de problemas cuando surgen errores durante el proceso de integración.

10.3. Point-to-point

El patrón de File Transfer Point-to-Point es un enfoque utilizado para transferir archivos entre dos puntos de manera directa, sin la necesidad de intermediarios adicionales. Este patrón es comúnmente utilizado en sistemas distribuidos donde la transferencia de archivos entre dos sistemas o nodos específicos es necesaria de manera eficiente y directa. Aquí tienes una descripción detallada del patrón:

Descripción:

1. Puntos de Transferencia (Transfer Points):
   - Son los dos extremos entre los cuales se realiza la transferencia de archivos.
   - Cada punto de transferencia puede ser un sistema informático, un servidor, un nodo de red, etc.
   - Los puntos de transferencia pueden estar ubicados en diferentes ubicaciones geográficas, redes o dominios administrativos.

2. Canal de Comunicación:
   - Se establece un canal de comunicación directo entre los dos puntos de transferencia para facilitar la transferencia de archivos.
   - Este canal puede ser una conexión de red, como una conexión TCP/IP o una conexión punto a punto dedicada.

3. Protocolo de Transferencia:
   - Se utiliza un protocolo de transferencia de archivos para definir el formato y las reglas para la transferencia de archivos entre los puntos de transferencia.
   - Los protocolos comunes incluyen FTP (File Transfer Protocol), SFTP (SSH File Transfer Protocol), SCP (Secure Copy Protocol), HTTP/S, entre otros.

4. Seguridad y Autenticación:
   - Se implementan medidas de seguridad y autenticación para proteger la integridad y la confidencialidad de los archivos transferidos.
   - Esto puede incluir el cifrado de datos, la autenticación de usuarios o sistemas, y el control de acceso a los archivos.

Flujo de trabajo típico:

1. El sistema emisor inicia una solicitud de transferencia de archivo al sistema receptor.
2. El sistema receptor acepta la solicitud y prepara el canal de comunicación para la transferencia.
3. El sistema emisor prepara el archivo para la transferencia y lo envía al sistema receptor a través del canal de comunicación.

4. El sistema receptor recibe el archivo y realiza las operaciones necesarias, como la validación de integridad, el procesamiento o el almacenamiento del archivo.
5. Una vez completada la transferencia, se envía una confirmación al sistema emisor para indicar que el archivo se ha transferido con éxito.

Ejemplo:

Supongamos que una empresa tiene dos sucursales ubicadas en diferentes ciudades y necesita transferir archivos de datos diariamente entre ellas. Para lograr esto, establecen una conexión VPN segura entre las dos sucursales y utilizan el protocolo SFTP para la transferencia de archivos. Cada día, el sistema en la sucursal A prepara los archivos de datos y los transfiere de manera segura a la sucursal B a través de la conexión VPN utilizando el protocolo SFTP. Una vez que se completa la transferencia, se envía una notificación de confirmación a la sucursal A para indicar que los archivos se han transferido correctamente.

Este es solo un ejemplo básico del patrón de File Transfer Point-to-Point en acción. Este patrón es útil cuando se necesita transferir archivos de manera eficiente y directa entre dos puntos específicos en un entorno distribuido, sin la necesidad de intermediarios adicionales.

10.4. One-to-many

El patrón de File Transfer One-to-Many se utiliza para transferir un archivo desde un origen a múltiples destinos de manera eficiente. Este patrón es común en situaciones donde se necesita distribuir el mismo archivo a múltiples receptores simultáneamente. Aquí tienes una descripción detallada del patrón:

Descripción:

1. Origen (Source):
   - Es el punto de inicio desde donde se inicia la transferencia del archivo.
   - El origen puede ser un sistema informático, un servidor, un dispositivo de almacenamiento, etc.

2. Destinos (Destinations):
   - Son los puntos de destino donde se desea transferir el archivo.
   - Los destinos pueden ser múltiples sistemas, servidores, dispositivos, etc.
   - Cada destino puede estar ubicado en diferentes ubicaciones geográficas, redes o dominios administrativos.

3. Canal de Comunicación:
   - Se establece un canal de comunicación entre el origen y cada uno de los destinos para facilitar la transferencia del archivo.
   - Este canal puede ser una conexión de red, como una conexión TCP/IP, una conexión punto a punto dedicada o un servicio de mensajería multicast.

4. Protocolo de Transferencia:
   - Se utiliza un protocolo de transferencia de archivos para definir el formato y las reglas para la transferencia del archivo desde el origen a los destinos.

- Los protocolos comunes incluyen FTP (File Transfer Protocol), SFTP (SSH File Transfer Protocol), SCP (Secure Copy Protocol), HTTP/S, entre otros.

5. Mecanismo de Distribución:
   - Se implementa un mecanismo de distribución para garantizar que el archivo se entregue de manera eficiente a todos los destinos.
   - Esto puede incluir técnicas como la transmisión simultánea a múltiples destinos, la fragmentación del archivo para reducir la congestión de red, el uso de caches intermedios, entre otros.

Flujo de trabajo típico:

1. El sistema origen inicia la transferencia del archivo a los destinos.
2. Se establecen canales de comunicación individuales entre el origen y cada uno de los destinos.
3. El sistema origen prepara el archivo para la transferencia y lo envía a cada uno de los destinos a través de los canales de comunicación establecidos.
4. Cada destino recibe el archivo y realiza las operaciones necesarias, como la validación de integridad, el procesamiento o el almacenamiento del archivo.

Ejemplo:

Supongamos que una empresa necesita distribuir un archivo de actualización de software a múltiples sucursales ubicadas en diferentes ciudades. Utilizan un servidor central como origen y establecen conexiones seguras (por ejemplo, VPN) con cada sucursal como destino. El servidor central envía el archivo de actualización a través de canales de comunicación seguros a todas las sucursales simultáneamente utilizando un protocolo de transferencia como SCP. Cada sucursal recibe el archivo de actualización y lo aplica en sus sistemas locales.

Este es solo un ejemplo básico del patrón de File Transfer One-to-Many en acción. Este patrón es útil cuando se necesita distribuir un archivo desde un origen a múltiples destinos de manera eficiente y simultánea.

10.5. File Transport

El patrón de File Transport es un enfoque utilizado para transferir archivos entre sistemas o componentes en un entorno distribuido. Este patrón se utiliza cuando se necesitan transferir grandes volúmenes de datos o archivos entre sistemas de manera eficiente y confiable. Aquí tienes una descripción detallada del patrón:

Descripción:

1. Origen (Source):
   - Es el sistema o componente que genera o posee el archivo que se va a transferir.
   - Puede ser un sistema informático, un servidor, un dispositivo de almacenamiento, etc.

2. Destino (Destination):
   - Es el sistema o componente al que se envía el archivo.
   - Puede ser un sistema informático, un servidor, un dispositivo de almacenamiento, etc.

3. Canal de Comunicación:
   - Se establece un canal de comunicación entre el origen y el destino para facilitar la transferencia del archivo.
   - Este canal puede ser una conexión de red, como una conexión TCP/IP, una conexión punto a punto dedicada o un servicio de mensajería.

4. Protocolo de Transferencia:
   - Se utiliza un protocolo de transferencia de archivos para definir el formato y las reglas para la transferencia del archivo desde el origen al destino.
   - Los protocolos comunes incluyen FTP (File Transfer Protocol), SFTP (SSH File Transfer Protocol), SCP (Secure Copy Protocol), HTTP/S, entre otros.

5. Mecanismo de Transferencia:
   - Se implementa un mecanismo de transferencia para garantizar que el archivo se transfiera de manera eficiente y confiable.
   - Esto puede incluir técnicas como la fragmentación del archivo para reducir la congestión de red, la compresión de datos para reducir el tamaño del archivo, el cifrado de datos para garantizar la seguridad de la transferencia, entre otros.

Flujo de trabajo típico:

1. El sistema origen inicia la transferencia del archivo al sistema destino.
2. Se establece un canal de comunicación entre el origen y el destino para facilitar la transferencia del archivo.
3. El sistema origen prepara el archivo para la transferencia y lo envía al sistema destino a través del canal de comunicación establecido.
4. El sistema destino recibe el archivo y realiza las operaciones necesarias, como la validación de integridad, el procesamiento o el almacenamiento del archivo.

Ejemplo:

Supongamos que una empresa necesita transferir archivos de datos generados por un sistema de producción a un sistema de análisis en tiempo real para su procesamiento y análisis. Utilizan un protocolo de transferencia como HTTP para transferir los archivos de producción al sistema de análisis a través de una conexión de red segura. Una vez recibidos, el sistema de análisis procesa los archivos y genera informes en tiempo real basados en los datos recibidos.

Este es solo un ejemplo básico del patrón de File Transport en acción. Este patrón es útil cuando se necesitan transferir grandes volúmenes de datos o archivos entre sistemas de manera eficiente y confiable en un entorno distribuido.

10.6. Shared File System

El patrón de Shared File System (Sistema de Archivos Compartidos) se utiliza para transferir archivos entre sistemas o componentes que comparten un mismo sistema de archivos. En este patrón, varios sistemas tienen acceso compartido a un sistema de archivos común, lo que les permite leer y escribir archivos de manera colaborativa. Aquí tienes una descripción detallada del patrón:

Descripción:

1. Sistema de Archivos Compartidos:

- Es un sistema de almacenamiento de archivos que puede ser accedido y compartido por varios sistemas o componentes en una red.
- Los sistemas de archivos compartidos pueden estar implementados en un servidor de archivos dedicado o en un almacenamiento en red (NAS) que proporciona acceso compartido a través de protocolos como NFS (Network File System), SMB (Server Message Block) o FTP (File Transfer Protocol).

2. Origen y Destino:
   - Los sistemas que participan en la transferencia de archivos actúan tanto como origen como destino de los archivos.
   - Pueden ser sistemas informáticos, servidores, dispositivos de almacenamiento, etc., que tienen acceso al sistema de archivos compartidos.

3. Acceso Concurrente:
   - Varios sistemas pueden acceder al mismo archivo de manera simultánea para leer o escribir datos.
   - Esto permite una transferencia de archivos rápida y eficiente entre los sistemas que comparten el mismo sistema de archivos.

4. Mecanismo de Sincronización:
   - Puede ser necesario implementar un mecanismo de sincronización para garantizar la coherencia y la integridad de los datos entre los sistemas que acceden al sistema de archivos compartidos.
   - Esto puede incluir técnicas como el bloqueo de archivos, la sincronización de datos en tiempo real o periódica, y la detección y resolución de conflictos.

Flujo de trabajo típico:

1. El sistema origen guarda el archivo en el sistema de archivos compartidos.
2. El sistema destino accede al archivo desde el sistema de archivos compartidos y lo lee o lo procesa según sea necesario.
3. Si es necesario, el sistema destino puede realizar modificaciones en el archivo y guardar los cambios de nuevo en el sistema de archivos compartidos.
4. Otros sistemas que necesiten acceder al mismo archivo pueden hacerlo de manera simultánea y colaborativa a través del sistema de archivos compartidos.

Ejemplo:

Supongamos que una empresa tiene varios servidores web distribuidos geográficamente que necesitan acceder a los mismos archivos de contenido estático, como imágenes, videos o archivos de código fuente. Utilizan un sistema de archivos compartidos basado en NFS para almacenar estos archivos, permitiendo que todos los servidores web accedan y sirvan los mismos archivos de manera rápida y eficiente a través del sistema de archivos compartidos.

Este es solo un ejemplo básico del patrón de Shared File System en acción. Este patrón es útil cuando se necesitan transferir archivos entre sistemas que comparten un mismo sistema de archivos, proporcionando un acceso compartido y concurrente a los datos en un entorno distribuido.

10.7. Event-Driven File Processing

El patrón Event-Driven File Processing es un patrón de diseño utilizado para manejar y procesar archivos de manera automática en respuesta a eventos detectados, en lugar de en intervalos predefinidos o mediante solicitudes manuales. Este enfoque permite que los sistemas respondan de manera más dinámica y eficiente a las condiciones cambiantes y a la carga de trabajo, ideal para entornos donde los archivos se generan o modifican a ritmos variables.

Descripción:

1. Detectores de Eventos (Event Detectors):
- Estos son mecanismos o componentes que monitorean fuentes de datos, como directorios de archivos, para detectar eventos específicos como la creación, modificación o eliminación de archivos.
- Pueden estar configurados para observar continuamente o para realizar comprobaciones en intervalos regulares.

2. Colas de Eventos (Event Queues):
- Cuando un evento es detectado, se genera un mensaje de evento que se envía a una cola de eventos.
- La cola de eventos sirve como buffer y gestor de prioridades, asegurando que los eventos sean procesados en el orden en que llegaron o según las prioridades establecidas.

3. Procesadores de Eventos (Event Processors):
- Estos son sistemas o componentes diseñados para manejar y procesar eventos.
- Pueden ejecutar una variedad de operaciones en los archivos, como transformación de datos, validación, carga a sistemas de bases de datos, o incluso activar otros flujos de trabajo.

4. Logística de Procesamiento:
- Define cómo se manejan y se escalan los eventos y los procesos asociados a ellos, asegurando que el sistema pueda manejar picos de actividad sin degradar el rendimiento.

5. Seguridad y Manejo de Errores:

- Implementación de medidas de seguridad para proteger los datos durante el procesamiento, como cifrado de datos y control de acceso basado en roles.
- Mecanismos de manejo de errores robustos para asegurar que cualquier fallo en el procesamiento de un archivo no detenga o afecte el procesamiento de otros eventos.

Flujo de trabajo típico:

1. Detección de Eventos: Un detector de eventos identifica la creación de un nuevo archivo en un directorio específico y genera un mensaje de evento.
2. Enrutamiento del Evento: El mensaje de evento se envía a una cola donde se ordena para su procesamiento.
3. Procesamiento del Evento: Un procesador de eventos toma el mensaje de la cola, procesa el archivo según las reglas definidas (p.ej., extracción de datos, transformación), y realiza las acciones necesarias.

4. Post-Procesamiento: Después de procesar el archivo, se pueden realizar acciones adicionales como el almacenamiento del resultado en una base de datos, notificaciones, o la activación de otros flujos de trabajo.
5. Confirmación de Procesamiento: Una vez completado el proceso, se envía una confirmación y se actualiza el estado del sistema.

Ejemplo:

Imagine una aplicación que gestiona las transacciones de venta. Cada vez que se completa una venta, se genera un archivo de resumen en un directorio. Un detector de eventos monitorea este directorio y, al detectar un nuevo archivo, inicia el proceso: el archivo es enviado a la cola, luego un procesador lee el archivo, extrae la información relevante, y actualiza la base de datos de inventario en tiempo real. Este enfoque dinámico permite que la empresa tenga siempre información actualizada de sus inventarios sin intervención manual.

Esta descripción detalla cómo funciona el patrón de procesamiento de archivos basado en eventos y cómo puede ser aplicado en escenarios prácticos, proporcionando claridad y profundidad en cada componente y paso del proceso.

11. Messaging

Los modelos de integración basados en mensajeria ofrecen una serie de beneficios clave que pueden transformar la manera en que las empresas manejan la comunicación entre aplicaciones y servicios. Estos beneficios incluyen:

1. Comunicación desacoplada: La mensajería permite una comunicación desacoplada entre aplicaciones, eliminando la necesidad de que los servicios estén disponibles inmediatamente uno para el otro. Esto se logra mediante la introducción de un intermediario, como una cola de mensajes, que almacena temporalmente los mensajes hasta que el receptor está listo para procesarlos. Este enfoque reduce la necesidad de código de procesamiento de errores por parte de los desarrolladores de aplicaciones.

2. Entrega de mensajes confiable: Garantiza que la información crítica para el negocio no se pierda ni se duplique, y que los mensajes se entreguen una sola vez al destinatario. Esto elimina la necesidad de que la aplicación implemente lógica para evitar duplicación o pérdida de mensajes.

3. Seguridad: La infraestructura de mensajería permite configurar la encriptación de los mensajes en tránsito y en reposo, lo que alivia a los desarrolladores de la carga de implementar estas medidas de seguridad. Esto asegura una comunicación segura end-to-end, que es superior a las soluciones de encriptación más simplistas usadas en llamadas síncronas.

4. Gestión de carga de trabajo: En situaciones de alto volumen de solicitudes, las aplicaciones pueden saturarse y dejar de responder. Los servidores de mensajería actúan como un búfer, permitiendo que el trabajo se almacene temporalmente hasta que la aplicación esté lista para procesarlo. Esto evita que las aplicaciones se sobrecarguen, permitiendo en su lugar que tomen el trabajo cuando puedan procesarlo, simplificando significativamente la gestión de la carga de trabajo a través de la aplicación.

11.1. Buenas prácticas para nombramiento de Colas

Estándar de nombramiento y manejo de colas para el patrón de mensajería (Messaging) que incluye colas para errores de procesamiento:

Estándar de Nombramiento de Colas:

1. Cola Principal:
    - Nombre: `main_queue` (o cualquier otro nombre descriptivo).
    - Descripción: Esta es la cola principal utilizada para procesar mensajes y coordinar la comunicación entre diferentes componentes del sistema.

2. Cola de Errores:
    - Nombre: `error_queue` (o cualquier otro nombre descriptivo).
    - Descripción: Esta cola se utiliza para almacenar mensajes que no pudieron ser procesados debido a errores durante el procesamiento.
    - Propósito: Proporciona un mecanismo para gestionar y abordar los errores de manera asíncrona, permitiendo una fácil identificación y resolución de problemas.

Manejo de Colas:

1. Envío de Mensajes:
    - Los mensajes deben ser enviados a la cola principal (`main_queue`) para su procesamiento normal.
    - En caso de que ocurra un error durante el procesamiento, el mensaje debe ser movido a la cola de errores (`error_queue`) para su posterior análisis y resolución.

2. Reintento de Procesamiento:
    - Implementar un mecanismo de reintento de procesamiento para los mensajes en la cola de errores.
    - Establecer una política de reintento que determine el número de intentos y el intervalo entre intentos para cada mensaje.

3. Gestión de Errores:
    - Implementar un proceso de gestión de errores que monitoree la cola de errores y tome medidas para abordar los mensajes que no se pudieron procesar.
    - Registrar información detallada sobre los errores, incluyendo el contenido del mensaje, la causa del error y cualquier información adicional relevante.

    Ejemplo de Implementación:

    - messaging_system/
      - main_queue/
        - message_1.json
        - message_2.xml
      - error_queue/
        - error_message_1.json
        - error_message_2.xml

En este ejemplo, la carpeta `main_queue` contiene mensajes enviados para su procesamiento normal, mientras que la carpeta `error_queue` contiene mensajes que no pudieron ser

procesados y requieren atención adicional. Este enfoque permite una fácil identificación y gestión de los errores de procesamiento en el sistema de mensajería, mejorando la fiabilidad y la capacidad de recuperación del sistema.

11.2. Point-to-point

El patrón de mensajería Point-to-Point (Punto a Punto) se utiliza en sistemas distribuidos para permitir la comunicación entre dos partes de manera directa y unidireccional. En este patrón, un remitente envía un mensaje a un destinatario específico a través de una cola de mensajes sin la necesidad de intermediarios adicionales. A continuación se encuentra una descripción detallada del patrón:

Descripción:

1. Remitente (Sender):
   - Es la parte que inicia el envío del mensaje.
   - Puede ser cualquier componente, aplicación o sistema que genere un mensaje para ser enviado al destinatario.

2. Destinatario (Receiver):
   - Es la parte que recibe el mensaje enviado por el remitente.
   - Puede ser cualquier componente, aplicación o sistema que esté configurado para escuchar y procesar mensajes de la cola de mensajes.

3. Cola de Mensajes (Message Queue):
   - Es un componente del sistema de mensajería que actúa como intermediario para el almacenamiento temporal de mensajes.
   - El remitente coloca el mensaje en la cola de mensajes, y el destinatario lo retira de la cola para su procesamiento.

4. Canal de Comunicación:
   - Se establece un canal de comunicación unidireccional entre el remitente y la cola de mensajes, y otro canal entre la cola de mensajes y el destinatario.
   - Los mensajes se transmiten a través de estos canales de comunicación de manera asincrónica.

Flujo de trabajo típico:

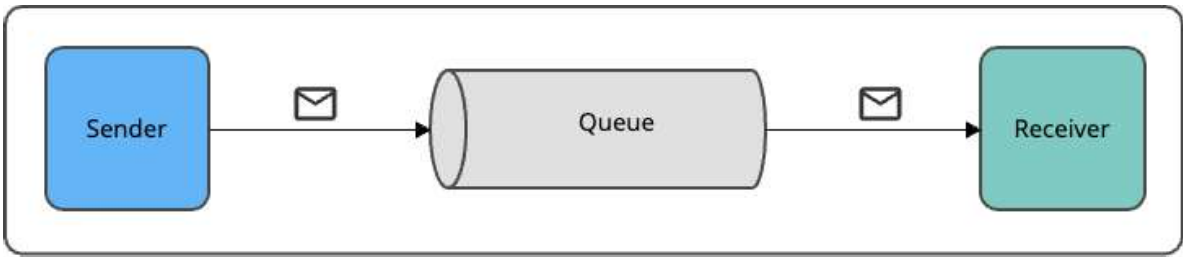
1. El remitente crea un mensaje y lo coloca en la cola de mensajes.
2. La cola de mensajes almacena el mensaje temporalmente hasta que el destinatario esté listo para procesarlo.
3. El destinatario retira el mensaje de la cola y lo procesa según sea necesario.
4. Una vez que el mensaje ha sido procesado, el destinatario puede enviar una respuesta al remitente a través de otro canal de comunicación, si es necesario.

Ejemplo:

Supongamos que una aplicación de comercio electrónico genera un mensaje cada vez que se realiza una nueva compra. Este mensaje contiene detalles sobre la compra, como el ID del pedido, los productos comprados y la información del cliente. El mensaje se coloca en una cola de mensajes Point-to-Point. Luego, un servicio de procesamiento de pedidos retira el mensaje de la cola, procesa la orden y actualiza el inventario. En este caso, la cola de mensajes facilita

la comunicación asincrónica entre la aplicación de comercio electrónico y el servicio de procesamiento de pedidos de manera eficiente y confiable.

Este es solo un ejemplo básico del patrón de mensajería Point-to-Point en acción. Este patrón es útil cuando se necesita una comunicación unidireccional y directa entre dos partes en un sistema distribuido, sin la necesidad de intermediarios adicionales.



Comunicación Punto a Punto usando colas de mensajería

11.3. Publish subscriber

El patrón de mensajería Publish/Subscribe (Publicar/Suscribir) se utiliza en sistemas distribuidos para permitir la comunicación entre múltiples partes de manera flexible y desacoplada. En este patrón, los remitentes (editores) envían mensajes a un tema (topic), y los destinatarios (suscriptores) reciben los mensajes de los temas a los que están suscritos. A continuación se encuentra una descripción detallada del patrón:

Descripción:

1. Editor (Publisher):
   - Es la parte que publica o envía mensajes a un tema específico.
   - Puede ser cualquier componente, aplicación o sistema que genere mensajes para ser distribuidos a los suscriptores.

2. Suscriptor (Subscriber):
   - Es la parte que recibe mensajes de los temas a los que está suscrito.
   - Puede ser cualquier componente, aplicación o sistema que esté configurado para escuchar y procesar mensajes de los temas de interés.

3. Tema (Topic):
   - Es una canal de comunicación abstracto que actúa como un punto central de distribución para mensajes relacionados.
   - Los editores publican mensajes en temas específicos, y los suscriptores reciben mensajes de los temas a los que están suscritos.

4. Canal de Comunicación:
   - Se establece un canal de comunicación entre los editores y los temas, y otro canal entre los temas y los suscriptores.
   - Los mensajes se transmiten a través de estos canales de comunicación de manera asincrónica.

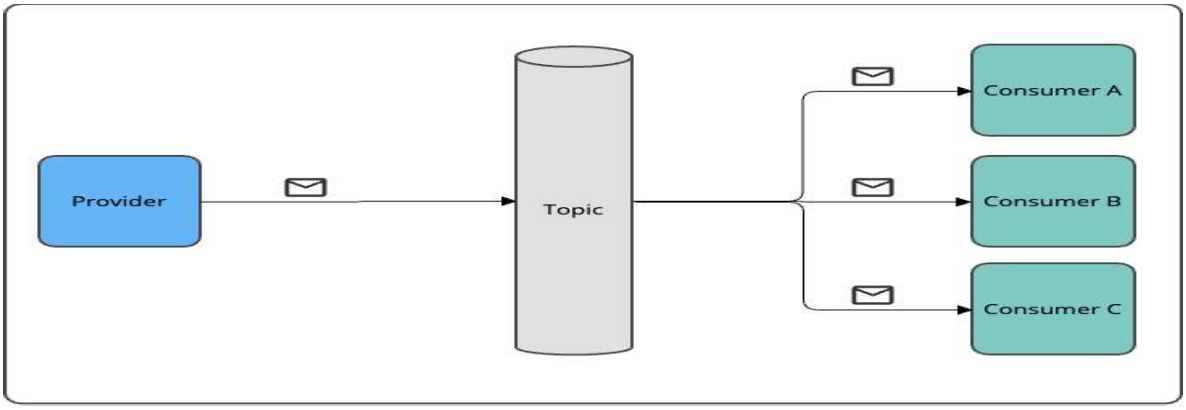
Flujo de trabajo típico:

1. Los editores publican mensajes en temas específicos.
2. Los mensajes se entregan a todos los suscriptores que estén suscritos a los temas relevantes.
3. Los suscriptores reciben y procesan los mensajes de los temas a los que están suscritos.
4. Los suscriptores pueden suscribirse o cancelar su suscripción a temas en cualquier momento según sus necesidades.

Ejemplo:

Supongamos que una aplicación de noticias tiene múltiples secciones, como deportes, política y tecnología. Cada sección publica noticias en su propio tema respectivo. Los usuarios interesados en noticias deportivas se suscriben al tema "Deportes", mientras que los interesados en política se suscriben al tema "Política". Los editores publican noticias en los temas correspondientes, y los suscriptores reciben las noticias de los temas a los que están suscritos. Esto permite una distribución eficiente y desacoplada de las noticias a los usuarios según sus intereses específicos.

Este es solo un ejemplo básico del patrón de mensajería Publish/Subscribe en acción. Este patrón es útil cuando se necesita una comunicación flexible y desacoplada entre múltiples partes en un sistema distribuido, permitiendo a los editores y suscriptores comunicarse de manera eficiente sin conocerse entre sí directamente.



Comunicación publicar suscribir usando colas de mensajería

11.4. Event message

El patrón de mensajería de eventos (Event Message) se utiliza en sistemas distribuidos para notificar a los consumidores sobre eventos o cambios que ocurren en el sistema. En este patrón, los productores generan eventos y los envían como mensajes a un sistema de mensajería, y los consumidores suscriben o filtran estos eventos según sus necesidades. Aquí tienes una descripción detallada del patrón:

Descripción:

1. Productor (Producer):
   - Es la parte que genera eventos y los envía como mensajes al sistema de mensajería.
   - Puede ser cualquier componente, aplicación o sistema que genere eventos en el sistema.

2. Consumidor (Consumer):
   - Es la parte que recibe y procesa los eventos enviados por los productores.
   - Puede ser cualquier componente, aplicación o sistema que esté configurado para escuchar y procesar eventos del sistema.

3. Evento (Event):
   - Es una notificación de un cambio o una acción que ha ocurrido en el sistema.
   - Los eventos pueden contener información relevante sobre el cambio, como datos relacionados, una marca de tiempo y el tipo de evento.

4. Canal de Comunicación:
   - Se establece un canal de comunicación entre los productores y el sistema de mensajería, y otro canal entre el sistema de mensajería y los consumidores.
   - Los eventos se transmiten a través de estos canales de comunicación de manera asincrónica.

Flujo de trabajo típico:

1. Los productores generan eventos en respuesta a cambios o acciones en el sistema.
2. Los eventos se envían como mensajes al sistema de mensajería.
3. Los consumidores se suscriben a eventos específicos o filtran eventos según sus necesidades.
4. Los consumidores reciben los eventos y los procesan según sea necesario.
5. Los consumidores pueden tomar acciones adicionales en respuesta a los eventos recibidos, como actualizar datos, enviar notificaciones, o activar flujos de trabajo.

Ejemplo:

Supongamos que una aplicación de monitoreo de servidores genera eventos cuando ocurren fallos en los servidores. Estos eventos se envían como mensajes al sistema de mensajería, donde varios consumidores están suscritos para recibir notificaciones sobre fallos de servidores. Cada vez que se produce un fallo en un servidor, se genera un evento correspondiente y se envía a los consumidores suscritos, que pueden tomar acciones para resolver el problema, como alertar a los administradores del sistema o iniciar un proceso de recuperación automatizado.

Este es solo un ejemplo básico del patrón de mensajería de eventos en acción. Este patrón es útil cuando se necesita notificar a los consumidores sobre cambios o eventos en el sistema de manera eficiente y asincrónica, permitiendo una comunicación flexible y desacoplada entre productores y consumidores.

11.5. Channel adapter

El patrón de Channel Adapter (Adaptador de Canal) se utiliza en sistemas de integración para conectar canales de comunicación heterogéneos con sistemas de mensajería unificados. Este patrón permite que diferentes sistemas de mensajería o canales de comunicación se integren de manera transparente con una arquitectura de mensajería centralizada. Aquí tienes una descripción detallada del patrón:

Descripción:

1. Canal de Comunicación (Communication Channel):
   - Es el medio a través del cual se transmiten los datos o mensajes entre sistemas o componentes.
   - Puede ser un canal de red, una cola de mensajes, un archivo, una base de datos, un servicio web, etc.

2. Adaptador de Canal (Channel Adapter):
   - Es un componente que actúa como intermediario entre el canal de comunicación y el sistema de mensajería centralizado.
   - Transforma los datos recibidos del canal de comunicación en un formato compatible con el sistema de mensajería centralizado y viceversa.
   - Maneja la lógica específica del canal de comunicación, como la lectura y escritura de datos, la gestión de conexiones y la transformación de formatos.

3. Sistema de Mensajería Centralizado (Centralized Messaging System):
   - Es un sistema de mensajería unificado que proporciona capacidades de enrutamiento, transformación, filtrado y gestión de mensajes.
   - Permite que los diferentes sistemas y componentes se comuniquen de manera coherente y uniforme a través de adaptadores de canal.

Flujo de trabajo típico:

1. El adaptador de canal recibe datos del canal de comunicación en su formato nativo.
2. El adaptador de canal transforma los datos en un formato compatible con el sistema de mensajería centralizado.
3. Los datos transformados se envían al sistema de mensajería centralizado para su procesamiento adicional.
4. El sistema de mensajería centralizado puede realizar acciones adicionales sobre los datos, como enrutamiento, filtrado, transformación adicional, etc.
5. Si se requiere una respuesta, el sistema de mensajería centralizado puede enviar una respuesta al adaptador de canal.
6. El adaptador de canal transforma la respuesta en el formato requerido por el canal de comunicación y la envía de vuelta al canal.

Ejemplo:

Supongamos que una empresa utiliza una combinación de sistemas de mensajería internos, servicios web externos y archivos planos para intercambiar datos con sus socios comerciales. Para integrar estos diferentes canales de comunicación con su sistema de mensajería centralizado, implementan adaptadores de canal específicos para cada canal. Estos adaptadores de canal se encargan de traducir los datos entre los formatos específicos del canal y los formatos estándar utilizados por el sistema de mensajería centralizado, permitiendo una integración transparente y eficiente entre los sistemas heterogéneos.

Este es solo un ejemplo básico del patrón de Channel Adapter en acción. Este patrón es útil cuando se necesite integrar diferentes canales de comunicación con un sistema de mensajería centralizado, permitiendo una comunicación uniforme y coherente entre sistemas heterogéneos en un entorno de integración de aplicaciones.

11.6. Dead letter channel

El patrón de Dead Letter Channel (Canal de Mensajes Muertos) es una técnica utilizada en sistemas de mensajería para manejar mensajes que no se pueden procesar con éxito y que no tienen un destino válido. Este patrón proporciona un mecanismo para capturar y almacenar mensajes no entregables, lo que permite la identificación y el análisis de los problemas de procesamiento de mensajes. Aquí tienes una descripción detallada del patrón:

Descripción:

1. Mensaje no entregable (Undeliverable Message):
   - Es un mensaje que no puede ser procesado con éxito o entregado a su destino previsto.
   - Puede ser el resultado de una variedad de situaciones, como errores de procesamiento, falta de destino válido, mensajes malformados, etc.

2. Canal de Mensajes Muertos (Dead Letter Channel):
   - Es un canal de almacenamiento diseñado específicamente para capturar y retener mensajes no entregables.
   - Proporciona un repositorio centralizado donde los mensajes no entregables pueden ser almacenados temporalmente para su posterior análisis o resolución.
   - Los mensajes almacenados en el canal de mensajes muertos pueden ser examinados manualmente por los administradores del sistema o procesados automáticamente según las reglas predefinidas.

Funcionamiento típico:

1. Cuando un mensaje no puede ser entregado con éxito, se envía al canal de mensajes muertos en lugar de ser descartado.
2. Los mensajes almacenados en el canal de mensajes muertos pueden ser examinados periódicamente por los administradores del sistema para identificar y analizar los problemas de procesamiento.
3. Los administradores pueden tomar medidas correctivas según sea necesario, como corregir errores de configuración, ajustar las reglas de enrutamiento de mensajes, o realizar acciones manuales para procesar los mensajes no entregables.
4. Dependiendo de la implementación, los mensajes no entregables pueden ser procesados automáticamente según las reglas predefinidas, como el reenvío a un destino alternativo o la notificación de errores a los usuarios finales.

Ejemplo:

Supongamos que una aplicación de procesamiento de pedidos en una tienda en línea recibe un mensaje de confirmación de pedido de un proveedor, pero el sistema no puede procesar el mensaje debido a un error de formato en el mensaje recibido. En lugar de perder el mensaje, se envía al canal de mensajes muertos para su posterior análisis. Los administradores del sistema pueden examinar el mensaje, identificar el error de formato y corregirlo. Una vez corregido, el mensaje puede ser reprocesado correctamente.

Este es solo un ejemplo básico del patrón de Dead Letter Channel en acción. Este patrón es útil para garantizar que los mensajes no entregables sean capturados y manejados adecuadamente, lo que ayuda a mejorar la fiabilidad y la capacidad de recuperación de los sistemas de mensajería.

11.7. Invalid letter channel

El concepto de "Invalid Letter Channel" no es un patrón de mensajería establecido, pero podríamos interpretarlo como una práctica para manejar mensajes inválidos en un sistema de mensajería. A continuación, te presento una descripción de cómo podrías abordar este tema en un entorno de mensajería:

Canal de Mensajes Inválidos:

El Canal de Mensajes Inválidos es un componente del sistema de mensajería que se utiliza para manejar y procesar mensajes que no cumplen con los criterios de validación establecidos. Este canal actúa como un repositorio donde se almacenan temporalmente los mensajes inválidos para su posterior análisis y acción correctiva.

Funcionalidades Clave:

1. Recepción de Mensajes Inválidos:
   - Los mensajes que no pasan la validación inicial son dirigidos al Canal de Mensajes Inválidos en lugar de ser descartados.
   - Esto garantiza que los mensajes inválidos no se pierdan y puedan ser revisados y corregidos posteriormente.

2. Registro y Almacenamiento:
   - Los mensajes inválidos son registrados y almacenados en el Canal de Mensajes Inválidos junto con información adicional relevante, como la razón del rechazo y el remitente original.
   - Este registro proporciona una pista de auditoría de los mensajes inválidos y facilita el análisis posterior.

3. Análisis y Corrección:
   - Los mensajes almacenados en el Canal de Mensajes Inválidos son revisados por los administradores del sistema para identificar la causa del problema.
   - Se toman medidas correctivas para corregir los errores en los mensajes, como ajustar las reglas de validación, limpiar los datos o notificar a los remitentes sobre los problemas de formato.

4. Reprocesamiento o Descarte:
   - Una vez corregidos, los mensajes inválidos pueden ser reprocesados y dirigidos nuevamente al flujo de procesamiento principal.
   - Si un mensaje inválido no se puede corregir o no tiene sentido continuar procesándolo, puede ser descartado después de un período de retención determinado.

Ejemplo de Implementación:

Supongamos que un sistema de procesamiento de pedidos en una empresa de comercio electrónico recibe un mensaje de pedido con una estructura de datos incorrecta. En lugar de descartar el mensaje, se redirige al Canal de Mensajes Inválidos. Los administradores del sistema revisan regularmente el Canal de Mensajes Inválidos, identifican el problema de formato en el mensaje de pedido y ajustan las reglas de validación para garantizar que futuros mensajes cumplan con los criterios esperados. Una vez corregido el problema, el mensaje de pedido puede ser reprocesado correctamente.

En resumen, el Canal de Mensajes Inválidos es una práctica importante en sistemas de mensajería para garantizar la integridad y la fiabilidad de los datos, permitiendo que los mensajes inválidos sean identificados, analizados y corregidos de manera oportuna.

12. Patrones de Diseño Arquitectónico

12.1. Microservicios

- Descripción: Este patrón organiza la plataforma como un conjunto de servicios pequeños, autónomos y enfocados en funcionalidades específicas.
- Implementación:
    - Servicios para gestión de usuarios, certificados catastrales, mutaciones de propiedad, entre otros.
    - Cada servicio se despliega de forma independiente en contenedores Docker gestionados por Kubernetes.

12.2. API Gateway

- Descripción: Centraliza todas las interacciones del frontend (web y móvil) con los microservicios backend.
- Implementación:
    - Utilización de Kong Gateway para enrutar solicitudes, autenticar usuarios y aplicar políticas de seguridad.
    - Capacidad para realizar transformaciones de datos y gestionar cuotas de uso.
- Ventajas:
    - Simplifica la interacción entre clientes y microservicios.
    - Ofrece control centralizado de seguridad y acceso.

12.3. Event-Driven Architecture (De cara al Broker)

- Descripción: Permite la comunicación asincrónica entre microservicios mediante eventos, mejorando la capacidad de respuesta y resiliencia del sistema.
- Implementación:
    - Uso de RabbitMQ para gestionar colas de mensajes que notifican eventos como actualizaciones de trámites o generación de certificados.
- Ventajas:
    - Reduce la dependencia directa entre servicios.
    - Mejora la capacidad de manejar picos de carga.

12.4. Event-Driven Architecture (monitoreo de microservicios)

- Descripción: Proporciona resiliencia al sistema interrumpiendo temporalmente llamadas a servicios con fallos persistentes.
- Implementación:
    - Integración de Resilience4j en los microservicios para monitorear fallos y aplicar políticas de recuperación.
- Ventajas:
    - Evita el efecto cascada en caso de fallos.
    - Mejora la estabilidad general del sistema.

12.5. Stateless Services

- Descripción: Los servicios no almacenan datos ni estados localmente, lo que permite un escalado horizontal más eficiente.
- Implementación:
  - Persistencia de datos en PostgreSQL y almacenamiento temporal en Redis para reducir la latencia.
  - Los servicios se diseñan para manejar cada solicitud de forma aislada.
- Ventajas:
  - Simplifica el escalado y la recuperación en caso de fallos.

13. Database per Service

- Descripción: Cada microservicio gestiona su propia base de datos, asegurando independencia en el manejo de datos.
- Implementación:
  - PostgreSQL para servicios transaccionales.
  - MongoDB para almacenar datos no estructurados como logs o configuraciones dinámicas.
- Ventajas:
  - Incrementa la autonomía de los servicios.
  - Facilita la evolución independiente de cada módulo.

13.1. CQRS (Command Query Responsibility Segregation)

- Descripción: Separa las operaciones de lectura y escritura para optimizar el rendimiento.
- Implementación:
  - Comandos para mutaciones en datos catastrales y consultas optimizadas para reportes estadísticos y mapas geoespaciales.
- Ventajas:
  - Mejora la eficiencia de consultas complejas.
  - Incrementa la consistencia eventual en sistemas distribuidos.

13.2. Domain-Driven Design (DDD)

- Descripción: Estructura la arquitectura en torno a los dominios del negocio para reflejar sus conceptos clave.
- Implementación:
  - Definición de Bounded Contexts para dominios como Propiedad, Usuarios, y Certificados.
  - Uso de terminología del negocio directamente en los microservicios.
- Ventajas:
  - Mejora la alineación entre los desarrolladores y los objetivos del negocio.
  - Facilita la gestión de sistemas complejos.

13.3. Sidecar Pattern

- Descripción: Implementa contenedores auxiliares para manejar funcionalidades transversales como logging y monitoreo.
- Implementación:
  - Uso de contenedores Prometheus para métricas y Grafana Loki para logging centralizado.
- Ventajas:
  - Reutilización de funcionalidades transversales sin impactar los microservicios.
  - Aumenta la observabilidad del sistema.

14. Backend for Frontend (BFF)

- Descripción: Proporciona una capa intermedia específica para los clientes web y móvil.
- Implementación:
  - Creación de APIs optimizadas para React (web) y React Native (móvil).
  - Manejo de lógica específica como transformación de datos y personalización de respuestas.
- Ventajas:
  - Mejora la experiencia del usuario.
  - Reduce la complejidad del frontend.

14.1. Hexagonal Architecture (Ports and Adapters)

- Descripción: Separa el núcleo del negocio de las dependencias externas, lo que facilita los cambios y la extensibilidad.
- Implementación:
  - Adaptadores para interactuar con APIs externas, bases de datos y sistemas GIS.
  - Núcleo de negocio independiente de frameworks o tecnologías específicas.
- Ventajas:
  - Incrementa la testabilidad.
  - Permite cambios en las dependencias externas sin afectar el núcleo del sistema.

15. Decisiones de arquitectura

En el diseño y desarrollo de sistemas de software, la claridad y precisión de las definiciones de arquitectura juegan un papel fundamental en la comprensión y comunicación efectiva de los componentes, estructuras y relaciones que conforman un sistema. El presente capítulo, "Definiciones de Arquitectura", se erige como un pilar fundamental dentro de nuestra propuesta de Arquitectura de Referencia para la capa de interoperabilidad.

Para garantizar la coherencia y la exhaustividad en la definición de la arquitectura de interoperabilidad, es imperativo considerar una serie de atributos de calidad que sirvan como guía para evaluar y validar la idoneidad de nuestras decisiones arquitectónicas. Estos atributos, que abarcan desde la disponibilidad hasta el gobierno, actúan como criterios de evaluación que orientan el diseño hacia la consecución de sistemas robustos, flexibles y adaptativos.

A continuación, presentamos los ocho atributos de calidad que guiarán nuestra exploración y definición de la arquitectura de interoperabilidad:

- Disponibilidad: La capacidad del sistema para estar operativo y accesible cuando se requiere, minimizando el tiempo de inactividad y maximizando la confiabilidad de los servicios ofrecidos.

- Desempeño: La capacidad del sistema para responder de manera eficiente a las demandas del usuario, manteniendo tiempos de respuesta óptimos y gestionando de manera efectiva los recursos computacionales.

- Interoperabilidad: La capacidad del sistema para intercambiar datos y operar de manera integrada con otros sistemas, garantizando la comunicación efectiva entre diferentes plataformas y tecnologías.

- Seguridad: La capacidad del sistema para proteger los datos y recursos de accesos no autorizados, implementando mecanismos de autenticación, autorización y cifrado que salvaguarden la integridad y confidencialidad de la información.

- Observabilidad: La capacidad del sistema para monitorear y analizar su propio comportamiento, facilitando la detección temprana de fallos, la optimización del rendimiento y la toma de decisiones informadas.

- Mantenibilidad: La capacidad del sistema para ser modificado, actualizado y reparado de manera eficiente, minimizando el impacto de los cambios en la estructura y funcionalidad del sistema.

- Time-to-market: El tiempo necesario para llevar un producto al mercado desde su concepción inicial, optimizando los procesos de desarrollo y reduciendo los ciclos de entrega mediante prácticas ágiles y eficientes.

- Gobierno: El conjunto de políticas, procesos y procedimientos que rigen el desarrollo, despliegue y operación del sistema, garantizando la alineación con los objetivos estratégicos de la organización y el cumplimiento de los estándares regulatorios y de calidad.

Al considerar estos atributos de calidad como parte integral de nuestras definiciones de arquitectura, nos comprometemos a diseñar sistemas que no solo cumplan con los requisitos

funcionales, sino que también satisfagan las necesidades y expectativas de los usuarios, los equipos de desarrollo y las partes interesadas en su conjunto. En los siguientes apartados, profundizaremos en cada uno de estos atributos, delineando sus características, desafíos y consideraciones clave en el contexto de la arquitectura de interoperabilidad.

Para cada uno de los atributos de calidad mencionados anteriormente, describiremos a continuación un conjunto de tópicos a considerar. Por cada uno de ellos, se evaluarán potenciales decisiones arquitectónicas, basándonos en el estado del arte tanto del mercado.

15.1. Disponibilidad

Situación problema:	¿Qué medidas estamos tomando para garantizar que nuestros sistemas puedan seguir funcionando en caso de fallos inesperados?
Motivación:	La disponibilidad es fundamental para garantizar la continuidad del servicio y la satisfacción del usuario. Atender este atributo nos permite minimizar el impacto de posibles interrupciones en la operación del sistema.

15.2. Arquitectura de redundancia

Se considera para escenarios donde es necesario contar con redundancia de componentes críticos para garantizar la disponibilidad continua en caso de fallas.

Potenciales decisiones de arquitectura

- Escenario Nube: Implementar el sistema en múltiples zonas de disponibilidad dentro de la misma región de la nube para garantizar la redundancia y la disponibilidad en caso de fallos en una zona específica.

- Escenario Nube: Mantener réplicas de datos en múltiples regiones geográficas de la nube para asegurar la disponibilidad y la recuperación ante desastres en caso de pérdida de datos en una región.

- Escenario Nube: Desplegar el sistema en múltiples proveedores de servicios en la nube para evitar la dependencia de un solo proveedor y garantizar la redundancia en caso de fallo de una nube específica.

- Escenario Nube: Aprovechar servicios gestionados de la nube que ofrezcan garantías de alta disponibilidad, como bases de datos gestionadas, almacenamiento en la nube con replicación automática, y servicios de cómputo redundantes.

- Escenario OnPremise: Configurar clústeres de servidores que trabajen en conjunto para garantizar la disponibilidad continua del sistema, permitiendo la recuperación automática en caso de fallo de un nodo.

- Escenario OnPremise: Mantener réplicas de servidores y datos en múltiples ubicaciones geográficas, como sucursales o centros de datos secundarios, para asegurar la redundancia y la recuperación ante desastres.

- Escenario OnPremise: Configurar ambientes de despliegue en modo activo-activo o activo-pasivo, donde múltiples instancias del sistema estén disponibles para manejar la carga en tiempo real o para activarse en caso de fallo.

- Escenario OnPremise: Emplear hardware de alta disponibilidad, como dispositivos de almacenamiento RAID, servidores con fuentes de alimentación redundantes, y conmutadores de red en modo de alta disponibilidad.

15.3. Balanceo de carga

Se considera para escenarios donde es necesario distribuir equitativamente las solicitudes entre múltiples instancias o servidores para evitar puntos de congestión y mejorar la disponibilidad.

Potenciales decisiones de arquitectura

- Escenario Nube: Utilizar servicios de balanceadores de carga administrados proporcionados por el proveedor de la nube, como AWS Elastic Load Balancer (ELB), Google Cloud Load Balancer, o Azure Load Balancer. Estos servicios ofrecen configuración y escalado automáticos.

- Escenario Nube: Utilizar balanceadores de carga a nivel de aplicación que permiten direccionar el tráfico en función de características de capa 7, como el nombre del host o la ruta de URL. Ejemplos incluyen Application Load Balancer (ALB) de AWS, Network Load Balancer (NLB) de AWS y Global Load Balancer (GLB) de Google Cloud.

- Escenario Nube: Configurar grupos de autoescalado que automáticamente agreguen o eliminen instancias virtuales según la demanda de tráfico, ayudando a mantener un rendimiento óptimo y evitar la saturación de recursos.

- Escenario Nube: Utilizar servicios DNS que distribuyan el tráfico entre múltiples puntos de acceso en diferentes regiones o zonas de disponibilidad, redirigiendo a los usuarios al servidor más cercano geográficamente.

- Escenario Nube: Utilizar una red de distribución de contenido (CDN) para almacenar en caché y distribuir contenido estático a través de servidores ubicados estratégicamente en todo el mundo, reduciendo la carga en los servidores de origen y mejorando la velocidad de entrega.

- Escenario OnPremise: Implementar dispositivos de balanceo de carga dedicados que se encuentren dentro de la infraestructura local, como hardware de F5 Networks, Citrix ADC (anteriormente NetScaler), o de otros proveedores de red.

- Escenario OnPremise: Implementar soluciones de software de balanceo de carga en servidores locales, como NGINX, HAProxy, o soluciones de balanceo de carga integradas en plataformas de virtualización como VMware NSX.

- Escenario OnPremise: Configurar servicios DNS locales para distribuir el tráfico entre múltiples servidores en la red local, utilizando registros de equilibrio de carga (RR) o Round Robin para alternar entre múltiples direcciones IP.

- Escenario OnPremise: Configurar IPs virtuales (VIP) que representen un grupo de servidores y utilizar técnicas de enrutamiento para distribuir el tráfico de manera equitativa entre ellos.

- Escenario OnPremise: Implementar soluciones de redes definidas por software (SDN) que permitan configurar políticas de enrutamiento y balanceo de carga de manera centralizada y dinámica dentro del entorno local.

15.4. Desempeño

Situación problema:	¿Cómo estamos asegurando que nuestro sistema mantenga un rendimiento óptimo bajo diferentes cargas de trabajo?
Motivación:	El rendimiento afecta directamente la experiencia del usuario y la eficiencia operativa. Atender este atributo nos permite ofrecer un servicio rápido y confiable, mejorando la satisfacción del usuario y optimizando los recursos del sistema.

15.4.1. Escalabilidad vertical y horizontal

Se considera para escenarios donde es necesario diseñar el sistema para escalar de manera eficiente, añadiendo más instancias (escalabilidad horizontal) y/o aumentando los recursos de cada instancia (escalabilidad vertical)

Potenciales decisiones de arquitectura

- Escenario Nube: Utilizar servicios de la nube que permitan la escalabilidad automática de recursos, como instancias de máquinas virtuales (VMs), bases de datos y almacenamiento, para aumentar o disminuir automáticamente la capacidad en función de la demanda.

- Escenario Nube: Descomponer la aplicación en microservicios independientes que puedan ser escalados de manera individual, permitiendo una escalabilidad horizontal más granular y eficiente en función de los componentes que experimenten una mayor carga.

- Escenario Nube: Utilizar contenedores Docker y plataformas de orquestación como Kubernetes para implementar y gestionar aplicaciones de manera escalable y flexible, permitiendo la fácil replicación y distribución de cargas de trabajo.

- Escenario Nube: Aprovechar servicios gestionados de la nube, como AWS Auto Scaling, Google Cloud Autoscaler, o Azure Autoscale, que monitorean la carga de trabajo y automáticamente ajustan la capacidad de recursos para mantener el rendimiento óptimo.

- Escenario Nube: Utilizar una red de distribución de contenido (CDN) para almacenar en caché y distribuir contenido estático a través de servidores ubicados estratégicamente en todo el mundo, reduciendo la carga en los servidores de origen y mejorando la velocidad de entrega.

- Escenario OnPremise: Aumentar la capacidad de recursos en servidores físicos, como CPU, RAM y almacenamiento, para manejar un mayor volumen de carga de trabajo, aunque esto puede tener límites prácticos y costos asociados.

- Escenario OnPremise: Configurar clústeres de servidores y balanceadores de carga para distribuir la carga de trabajo entre múltiples nodos, permitiendo una escalabilidad horizontal dentro del entorno OnPremise.

- Escenario OnPremise: Descomponer la aplicación monolítica en microservicios independientes que puedan ser escalados de manera individual, permitiendo una escalabilidad horizontal más granular y eficiente.

- Escenario OnPremise: Utilizar tecnologías de virtualización, como VMware o Hyper-V, para consolidar múltiples cargas de trabajo en un número menor de servidores físicos, permitiendo una mejor utilización de los recursos y una mayor flexibilidad de escalabilidad.

- Escenario OnPremise: Utilizar sistemas de cacheo como Redis o Memcached para almacenar en memoria datos frecuentemente accedidos, reduciendo la carga en los sistemas de base de datos y mejorando el rendimiento global de la aplicación.

15.4.2. Caché

Se considera para escenarios donde es necesario implementar estrategias de almacenamiento en caché para reducir el tiempo de respuesta y la carga en los recursos

Potenciales decisiones de arquitectura

- Escenario Nube: Aprovechar servicios de caché gestionados proporcionados por el proveedor de la nube, como Amazon ElastiCache, Google Cloud Memorystore o Azure Cache for Redis, que ofrecen soluciones de caché completamente administradas y escalables.

- Escenario Nube: Desplegar una solución de caché distribuido utilizando tecnologías como Redis o Memcached, distribuyendo los datos en múltiples nodos para mejorar la disponibilidad y el rendimiento.

- Escenario Nube: Utilizar una red de distribución de contenido (CDN) para almacenar en caché contenido estático, como imágenes, archivos CSS y JavaScript, distribuyendo estos recursos a través de una red global de servidores cercanos a los usuarios finales para mejorar la velocidad de carga.

- Escenario Nube: Implementar una capa de caché a nivel de aplicación utilizando bibliotecas como Redisson o caffeine en el código de la aplicación, para almacenar en memoria resultados de consultas a bases de datos o resultados de cálculos costosos.

- Escenario Nube: Configurar un caché en la capa de API Gateway para almacenar temporalmente respuestas a solicitudes HTTP, reduciendo la carga en los servicios backend y mejorando el tiempo de respuesta para solicitudes repetidas.

- Escenario OnPremise: Configurar cache local en los servidores de la infraestructura OnPremise utilizando tecnologías como Redis o Memcached, para almacenar datos y resultados de consultas de forma rápida y eficiente.

- Escenario OnPremise: Implementar un servidor proxy de caché, como Squid, Varnish o Nginx, en la infraestructura local para almacenar en caché contenido web y reducir el tiempo de carga de páginas para los usuarios finales.

- Escenario OnPremise: Desarrollar una capa de caché en la aplicación utilizando bibliotecas como Ehcache o Guava Cache, para almacenar en memoria datos frecuentemente accedidos y reducir la necesidad de consultas a bases de datos.

- Escenario OnPremise: Configurar un sistema de caché distribuido entre los nodos de la infraestructura OnPremise utilizando tecnologías como Redis Cluster o Memcached, para mejorar la escalabilidad y la disponibilidad del caché.

- Escenario OnPremise: Utilizar un middleware de caché, como Apache Ignite o Hazelcast, para agregar una capa de caché entre las aplicaciones y las bases de datos, reduciendo la latencia de acceso a los datos y mejorando el rendimiento de las aplicaciones.

15.5. Interoperabilidad

Situación problema:	¿Cómo estamos facilitando la comunicación y colaboración entre nuestro sistema y otros sistemas externos?
Motivación:	La interoperabilidad es esencial para integrar nuestro sistema con otros sistemas y servicios, maximizando su utilidad y valor. Atender este atributo nos permite asegurar una comunicación fluida y efectiva en un entorno cada vez más interconectado.

15.6. Protocolos de comunicación

Se considera para escenarios donde es necesario definir los protocolos y formatos de intercambio de datos para permitir la comunicación entre sistemas heterogéneos.

Potenciales decisiones de arquitectura

- Utilizar protocolos de comunicación estándar como HTTP, HTTPS, REST, JSON-RPC o gRPC, que son ampliamente aceptados y compatibles con una amplia gama de tecnologías y plataformas en la nube.

- Desarrollar APIs bien documentadas y abiertas utilizando estándares como OpenAPI (anteriormente conocido como Swagger) o GraphQL, que facilitan la interoperabilidad al permitir a los clientes comprender y utilizar los servicios de manera consistente.

- Implementar sistemas de mensajería asincrónica utilizando protocolos como AMQP (Advanced Message Queuing Protocol) o MQTT (Message Queuing Telemetry Transport) para facilitar la comunicación entre componentes distribuidos de manera eficiente y confiable.

- Utilizar protocolos de integración empresarial como SOAP (Simple Object Access Protocol) o JMS (Java Message Service) cuando sea necesario interoperar con sistemas heredados o aplicaciones corporativas que requieran estos estándares.

- Implementar arquitecturas basadas en eventos utilizando protocolos como WebSockets o Server-Sent Events (SSE) para permitir una comunicación bidireccional en tiempo real entre clientes y servidores, lo que mejora la experiencia del usuario y la interoperabilidad.

- Desarrollar servicios web RESTful utilizando protocolos como HTTP y JSON para facilitar la comunicación entre aplicaciones y permitir la interoperabilidad a través de la web en la infraestructura local.

- Integrar sistemas heredados utilizando protocolos de comunicación legacy como TCP/IP, FTP (File Transfer Protocol) o JDBC (Java Database Connectivity) cuando sea necesario interoperar con sistemas antiguos que no admiten protocolos modernos.

15.7. APIs y estándares

Se considera para escenarios donde es necesario integración vertical, entre los principales estándares a considerar se encuentran:

- Utilizar estándares abiertos como REST (Representational State Transfer) y JSON (JavaScript Object Notation) para el diseño de APIs, lo que facilita la interoperabilidad al ser ampliamente aceptados y comprensibles.

- Proporcionar una documentación exhaustiva y clara para las APIs, utilizando herramientas como Swagger/OpenAPI o RAML, lo que permite a los desarrolladores comprender fácilmente cómo interactuar con los servicios ofrecidos y promueve la adopción.

- Utilizar protocolos de autenticación estándar como OAuth 2.0 para autorización y autenticación de usuarios en las APIs, lo que facilita la integración con sistemas de terceros y mejora la seguridad de la comunicación.

- Definir y utilizar esquemas de datos comunes, como JSON Schema o XML Schema, para estandarizar la estructura y el formato de los datos intercambiados a través de las APIs, lo que facilita la comprensión y la interoperabilidad entre sistemas.

15.8. Transformación de datos

Se considera para escenarios donde es necesario implementar mecanismos de transformación de datos para adaptar los formatos y estructuras entre sistemas incompatibles

Potenciales decisiones de arquitectura

- Escenario Nube: Aprovechar servicios de integración de datos en la nube, como AWS Glue, Google Cloud Dataflow o Azure Data Factory, que ofrecen capacidades de transformación de datos escalables y gestionadas en la nube.

- Escenario Nube: Desarrollar pipelines de datos utilizando herramientas de orquestación como Apache Airflow o AWS Step Functions, que permiten definir y ejecutar flujos de trabajo complejos para la transformación y carga de datos en entornos de nube.

- Escenario Nube: Utilizar servicios de ETL en la nube, como AWS Glue, Google Cloud Dataflow o Azure Data Factory, que proporcionan capacidades de extracción, transformación y carga de datos de manera escalable y automatizada.

- Escenario Nube: Desarrollar microservicios dedicados a la transformación de datos utilizando contenedores Docker y plataformas de orquestación como Kubernetes, lo que permite escalar y desplegar los servicios de manera eficiente en entornos de nube.

- Escenario Nube: Transformar los datos a formatos estándar como JSON o Avro durante el proceso de transformación, lo que facilita la interoperabilidad y el intercambio de datos entre diferentes sistemas y aplicaciones en la nube.

- Escenario OnPremise: Instalar y configurar servidores de integración de datos locales, como Apache NiFi o Talend Open Studio, que permiten diseñar y ejecutar flujos de trabajo de transformación de datos en la infraestructura OnPremise.

- Escenario OnPremise: Emplear herramientas ETL locales, como Informatica PowerCenter o IBM InfoSphere DataStage, que proporcionan capacidades de extracción, transformación y carga de datos en entornos OnPremise.

- Escenario OnPremise: Definir y aplicar estándares de datos internos para asegurar la coherencia y la consistencia en la transformación de datos en entornos OnPremise, lo que facilita la interoperabilidad y el intercambio de datos entre sistemas internos.

15.9. Seguridad

Situación problema:	¿Qué medidas estamos tomando para proteger los datos sensibles y prevenir accesos no autorizados a nuestro sistema?
Motivación:	La seguridad es crucial para proteger la confidencialidad, integridad y disponibilidad de la información. Atender este atributo nos permite salvaguardar la confianza del usuario y cumplir con regulaciones y estándares de seguridad.

15.10. Autenticación y autorización

Se considera para escenarios donde es necesario establecer mecanismos robustos de autenticación de usuarios (humanos y sistemas) y autorización de accesos a recursos sensibles.

Potenciales decisiones de arquitectura

- Escenario Nube: Aprovechar servicios de gestión de identidad y acceso en la nube, como AWS IAM (Identity and Access Management), Google Cloud IAM o Azure Active Directory, que proporcionan capacidades robustas de autenticación y autorización en entornos de nube.

- Escenario Nube: Configurar soluciones de SSO como OpenID Connect o SAML (Security Assertion Markup Language) para permitir que los usuarios inicien sesión una sola vez y accedan a múltiples servicios y aplicaciones en la nube sin necesidad de autenticarse repetidamente, mediante algún sistema que provea estos protocolos implementados, como por ejemplo Keycloak.

- Escenario Nube: Utilizar protocolos seguros de autenticación como OAuth 2.0 o OpenID Connect para permitir que los usuarios autoricen aplicaciones de terceros a acceder a sus datos sin compartir sus credenciales de acceso, mejorando la seguridad y la privacidad, mediante algún sistema que provea estos protocolos implementados, como por ejemplo Keycloak.

- Escenario Nube: Habilitar la autenticación multifactor (MFA) para los usuarios finales, que requiere la presentación de múltiples factores de autenticación, como contraseñas, códigos de verificación SMS o tokens de seguridad, para aumentar la seguridad de las cuentas.

- Escenario Nube: Establecer políticas de autorización basadas en roles (RBAC) que asignen permisos y privilegios a los usuarios en función de sus roles y responsabilidades dentro de la organización, lo que ayuda a limitar el acceso a los recursos sensibles en la nube.

- Escenario OnPremise: Configurar servidores de autenticación centralizada como LDAP (Lightweight Directory Access Protocol) o Active Directory en la infraestructura local para gestionar la autenticación de usuarios y dispositivos de manera centralizada.

- Escenario OnPremise: Establecer políticas de autorización granulares que definan los permisos de acceso a los recursos basados en roles, grupos y atributos de usuarios, lo que permite una gestión precisa y controlada de los accesos en entornos OnPremise.

15.11. Cifrado

Se considera para escenarios donde es necesario implementar técnicas seguras para proteger la confidencialidad e integridad de los datos durante su almacenamiento y transmisión.

Potenciales decisiones de arquitectura

- Escenario Nube: Utilizar HTTPS/TLS para cifrar los datos en tránsito entre clientes y servidores, asegurando que la comunicación entre aplicaciones y servicios en la nube esté protegida contra interceptaciones.

- Escenario Nube: Configurar el cifrado en reposo para datos almacenados en servicios de almacenamiento en la nube, utilizando tecnologías como AWS KMS (Key Management Service), Google Cloud KMS o Azure Key Vault, para cifrar automáticamente los datos antes de almacenarlos.

- Escenario Nube: Implementar cifrado de bases de datos utilizando servicios gestionados que soporten cifrado nativo, como Amazon RDS (Relational Database Service) con cifrado en reposo, Google Cloud SQL o Azure SQL Database.

- Escenario Nube: Utilizar servicios de gestión de claves en la nube, como AWS KMS, Google Cloud KMS o Azure Key Vault, para gestionar y rotar las claves de cifrado de manera segura y automatizada, asegurando que las claves estén protegidas y cumplan con las políticas de seguridad.

- Escenario Nube: Implementar cifrado a nivel de aplicación utilizando bibliotecas criptográficas como AWS Encryption SDK, Google Tink o Microsoft Cryptographic API, para cifrar datos sensibles antes de enviarlos a la capa de almacenamiento o base de datos.

- Escenario OnPremise: Usar el cifrado de datos en el motor de base de datos.

- Escenario OnPremise: Configurar un sistema de gestión de claves (KMS) interno utilizando herramientas como HashiCorp Vault o Thales CipherTrust, para gestionar, rotar y proteger claves de cifrado dentro de la infraestructura OnPremise.

- Escenario OnPremise: Implementar cifrado a nivel de aplicación utilizando bibliotecas criptográficas como OpenSSL, Bouncy Castle o Microsoft .NET Cryptography, para cifrar datos sensibles antes de almacenarlos o transmitirlos.

15.12. Gestión de identidades

Se considera para escenarios donde es necesario administrar de manera centralizada y segura las identidades de usuarios y servicios para garantizar el acceso adecuado a los recursos.

Potenciales decisiones de arquitectura

- Escenario Nube: Utilizar servicios de gestión de identidades proporcionados por el proveedor de la nube, como AWS IAM, Azure Active Directory (Azure AD) o Google Cloud Identity, que ofrecen control centralizado y capacidades de gestión de identidades y accesos.

- Escenario Nube: Configurar SSO utilizando protocolos estándar como SAML, OAuth 2.0 o OpenID Connect para permitir a los usuarios autenticarse una sola vez y acceder a múltiples aplicaciones y servicios en la nube sin necesidad de iniciar sesión repetidamente.

- Escenario Nube y OnPremise: Integrar el servicio de gestión de identidades en la nube con directorios de identidades externos, como Active Directory, LDAP o sistemas de identidad federada, para proporcionar una experiencia de inicio de sesión unificada y gestionar usuarios desde una ubicación centralizada.

- Escenario Nube: Habilitar MFA para añadir una capa adicional de seguridad al proceso de autenticación, requiriendo que los usuarios verifiquen su identidad mediante múltiples factores, como contraseñas, tokens físicos o aplicaciones de autenticación.

- Escenario Nube: Utilizar herramientas de gestión de identidades y accesos (IAM) para automatizar el ciclo de vida de las identidades, incluyendo la creación, actualización y eliminación de cuentas de usuario, basándose en políticas y flujos de trabajo definidos.

- Escenario OnPremise: Configurar y gestionar directorios de identidades locales, como Active Directory o LDAP, para proporcionar una infraestructura centralizada de

autenticación y autorización para todos los usuarios y dispositivos dentro de la organización.

- Escenario OnPremise: Desplegar soluciones de SSO locales, como Keycloak, para permitir a los usuarios autenticarse una sola vez y acceder a múltiples aplicaciones y servicios internos sin necesidad de iniciar sesión repetidamente

- Escenario OnPremise: Integrar diferentes sistemas de gestión de identidades y accesos (IAM) en la infraestructura OnPremise utilizando soluciones de federación de identidades, permitiendo una gestión unificada de usuarios a través de múltiples plataformas y aplicaciones.

- Escenario OnPremise: Implementar políticas estrictas de contraseñas y autenticación, como la complejidad de contraseñas, expiración regular y bloqueo de cuentas después de múltiples intentos fallidos, para mejorar la seguridad del acceso a los sistemas locales

15.13. Observabilidad

Situación problema:	¿Cómo estamos monitoreando y diagnosticando el rendimiento y la salud de nuestro sistema en tiempo real?
Motivación:	La observabilidad es clave para detectar y resolver problemas de manera proactiva, garantizando un funcionamiento óptimo del sistema. Atender este atributo nos permite identificar y abordar rápidamente cualquier anomalía o incidente que pueda surgir.

15.14. Registro y seguimientos de eventos

Se considera para escenarios donde es necesario implementar mecanismos para registrar eventos y actividades importantes dentro del sistema, facilitando la detección y resolución de problemas.

Potenciales decisiones de arquitectura

- Escenario Nube: Emplear servicios gestionados de registro en la nube, como Amazon CloudWatch, Google Cloud Logging o Azure Monitor, que ofrecen capacidades avanzadas de registro, monitoreo y análisis de eventos.

- Escenario Nube: Configurar una plataforma centralizada para el registro de eventos utilizando soluciones como ELK Stack (Elasticsearch, Logstash y Kibana), Fluentd o Splunk en su versión cloud, para recopilar, analizar y visualizar logs de manera eficiente

- Escenario Nube: Implementar soluciones de tracing distribuido como AWS X-Ray, Google Cloud Trace o Azure Application Insights para seguir el flujo de eventos y solicitudes a través de microservicios y componentes distribuidos en la nube, mejorando la visibilidad del rendimiento y el comportamiento de la aplicación

- Escenario Nube: Configurar alertas y monitoreo proactivo utilizando servicios como Amazon CloudWatch Alarms, Google Cloud Alerting o Azure Monitor Alerts, para detectar y responder a eventos inusuales o fallas en tiempo real

- Escenario Nube: Integrar servicios de registro con herramientas de observabilidad completas como Datadog, New Relic o Dynatrace, que proporcionan monitoreo, tracing, y análisis de logs en una sola plataforma

- Escenario OnPremise: Implementar una solución de registro centralizada como ELK Stack (Elasticsearch, Logstash y Kibana), Graylog o Splunk, para recopilar, almacenar y analizar logs provenientes de diversas fuentes dentro de la infraestructura OnPremise

- Escenario OnPremise: Configurar sistemas de monitoreo y alertas como Prometheus con Grafana, Nagios o Zabbix, para monitorear los eventos y el rendimiento de los sistemas locales y generar alertas en caso de detección de anomalías o fallas

- Escenario OnPremise: Utilizar soluciones de tracing distribuido como Jaeger o Zipkin para rastrear solicitudes y eventos a través de microservicios y componentes distribuidos dentro de la infraestructura OnPremise, mejorando la visibilidad y el diagnóstico de problemas

- Escenario OnPremise: Establecer políticas de segregación y retención de logs para asegurar que los datos relevantes se almacenen de manera segura y cumplan con las políticas de retención de la organización, utilizando soluciones de almacenamiento de logs locales.

- Escenario OnPremise: Configurar herramientas de análisis de logs y dashboards personalizados con herramientas como Grafana o Kibana, para proporcionar visualizaciones en tiempo real y análisis detallados de los eventos registrados

15.14.1. Monitoreo de métricas de rendimiento

Se considera para escenarios donde es necesario recopilar y analizar métricas de rendimiento como tiempo de respuesta, uso de recursos y errores para identificar áreas de mejora y optimización.

Potenciales decisiones de arquitectura

- Escenario Nube: Utilizar servicios gestionados de monitoreo de rendimiento, como Amazon CloudWatch, Google Cloud Monitoring (anteriormente Stackdriver) o Azure Monitor, para recopilar, visualizar y alertar sobre métricas de rendimiento de los recursos y aplicaciones en la nube

- Escenario Nube: Emplear soluciones de monitoreo del rendimiento de aplicaciones (APM) como New Relic, Datadog, Dynatrace o AWS X-Ray para obtener una visión detallada del rendimiento de las aplicaciones, incluyendo métricas de tiempo de respuesta, uso de recursos y seguimiento de transacciones.
-

- Escenario Nube: Integrar servicios de telemetría como AWS CloudTrail, Google Cloud Trace o Azure Monitor Logs para capturar y analizar datos de telemetría y eventos, proporcionando una visión completa del rendimiento y comportamiento de la infraestructura

- Escenario Nube: Crear dashboards personalizados utilizando herramientas como Grafana, integradas con servicios de monitoreo en la nube, para visualizar métricas clave de rendimiento en tiempo real y facilitar la toma de decisiones basada en datos

- Escenario Nube: Configurar alertas inteligentes basadas en umbrales y anomalías utilizando Amazon CloudWatch Alarms, Google Cloud Monitoring Alerting o Azure Monitor Alerts, para notificar a los equipos de operaciones sobre problemas de rendimiento antes de que afecten a los usuarios finales.

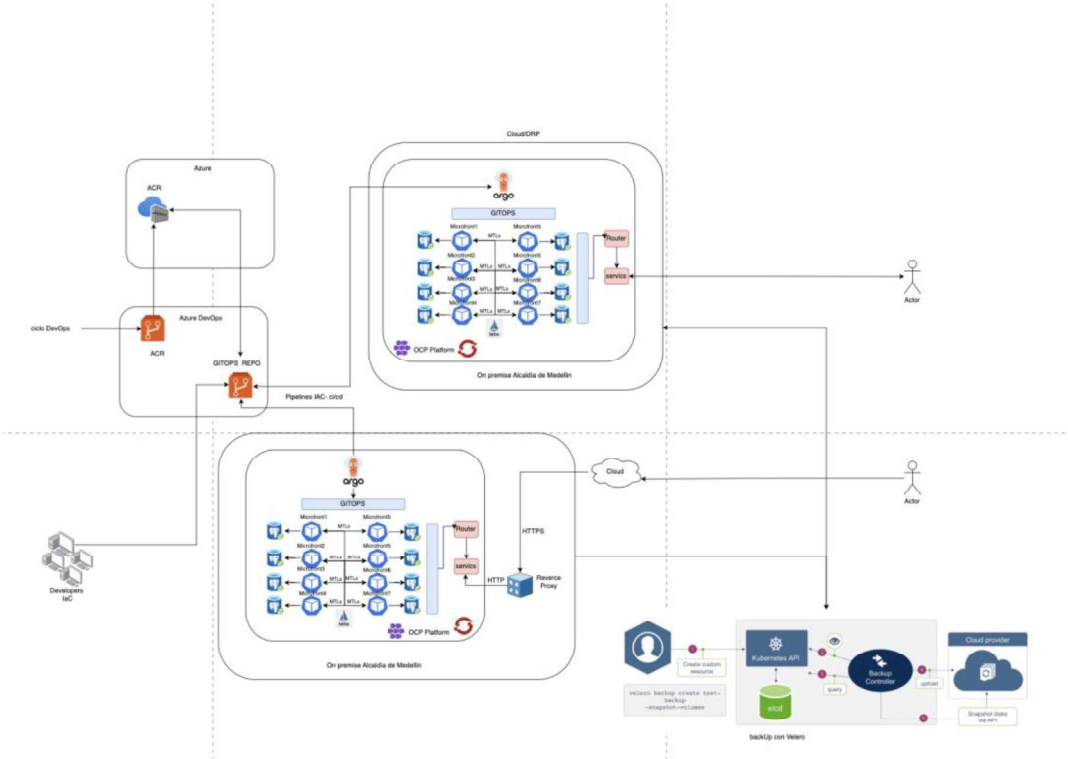
- Escenario OnPremise: Implementar herramientas de monitoreo de infraestructura como Prometheus con Grafana, Nagios o Zabbix para recopilar, almacenar y visualizar métricas de rendimiento de servidores, redes y aplicaciones en entornos OnPremise

- Escenario OnPremise: Desplegar agentes de recolección de métricas como Collectd o Telegraf en servidores y dispositivos para recopilar datos de rendimiento y enviarlos a una plataforma centralizada para su análisis y visualización

- Escenario OnPremise: Configurar sistemas de telemetría locales utilizando herramientas como Jaeger o Zipkin para rastrear y analizar el flujo de solicitudes y eventos, proporcionando una visión completa del rendimiento y comportamiento de las aplicaciones

15.15. Mantenibilidad

Situación problema:	¿Qué tan fácil es realizar cambios en nuestro sistema sin causar efectos no deseados o tiempos de inactividad?
Motivación:	La mantenibilidad asegura la capacidad de adaptación y evolución del sistema a lo largo del tiempo. Atender este atributo nos permite reducir el costo y la complejidad de mantener y actualizar el sistema, facilitando su mantenimiento a largo plazo.



16. Automatización de pruebas y despliegue usando DevSecOps

Para este punto se construirá un documento aparte con el detalle del proceso de DevSecOps para catastro

16.1. Time-To-Market

Situación problema:	¿Cuánto tiempo nos toma llevar nuevas funcionalidades o actualizaciones al mercado?
Motivación:	El tiempo de llegada al mercado es crítico para mantener la competitividad y satisfacer las demandas del usuario. Atender este atributo nos permite acelerar el ciclo de desarrollo y despliegue, maximizando la velocidad y eficiencia en la entrega de valor al usuario.

16.2. Iteraciones rápidas de desarrollo

Se considera para escenarios donde es necesario adoptar metodologías ágiles como Scrum o Kanban para acelerar el ciclo de desarrollo y entrega de software.

Potenciales decisiones de arquitectura

- Escenario Nube: Adoptar metodologías ágiles como Scrum o Kanban para organizar el trabajo en sprints cortos, iterativos y enfocados en la entrega continua de valor

- Escenario Nube: Utilizar servicios de CI/CD en la nube como AWS CodePipeline, Google Cloud Build o Azure DevOps para automatizar la construcción, prueba y despliegue de aplicaciones, permitiendo iteraciones rápidas y despliegues frecuentes

- Escenario Nube: Diseñar la arquitectura de la aplicación en microservicios, que pueden desarrollarse, desplegarse y escalarse de forma independiente, permitiendo iteraciones más rápidas y una mayor flexibilidad en el desarrollo

- Escenario Nube: Utilizar contenedores Docker y plataformas de orquestación como Kubernetes para empaquetar y desplegar aplicaciones rápidamente, facilitando la gestión de dependencias y la escalabilidad

- Escenario Nube: Implementar IaaS utilizando herramientas como AWS CloudFormation, Terraform o Azure Resource Manager, para automatizar el aprovisionamiento y gestión de la infraestructura, reduciendo los tiempos de configuración y permitiendo despliegues rápidos y consistentes.

- Escenario Nube: Utilizar técnicas como pruebas A/B y feature toggles para desplegar nuevas funcionalidades de manera controlada, permitiendo iteraciones rápidas y la recolección de feedback en tiempo real

- Escenario OnPremise: Utilizar herramientas de CI/CD como Jenkins, GitLab CI, o Bamboo para automatizar la construcción, prueba y despliegue de aplicaciones en el entorno OnPremise, permitiendo iteraciones rápidas y despliegues frecuentes

- Escenario OnPremise: Utilizar contenedores Docker y plataformas de orquestación como Kubernetes o OpenShift para empaquetar y desplegar aplicaciones rápidamente en la infraestructura OnPremise, facilitando la gestión de dependencias y la escalabilidad

- Escenario OnPremise: Automatizar las pruebas unitarias, de integración y de aceptación utilizando herramientas como JUnit, Selenium o Cucumber, y configurar pipelines de despliegue automatizado para reducir los tiempos de entrega y mejorar la calidad del software

16.3. Automatización de procesos

Se considera para escenarios donde es necesario implementar herramientas y prácticas de automatización para agilizar tareas repetitivas como pruebas, compilación y despliegue.

Potenciales decisiones de arquitectura

- Escenario Nube: Utilizar servicios gestionados de CI/CD como AWS CodePipeline, Google Cloud Build, o Azure DevOps para automatizar la construcción, prueba y despliegue de aplicaciones, facilitando iteraciones rápidas y despliegues frecuentes

- Escenario Nube: Implementar IaC utilizando herramientas como AWS CloudFormation, Terraform o Azure Resource Manager para automatizar el aprovisionamiento y la gestión de la infraestructura, asegurando configuraciones consistentes y repetibles

- Escenario Nube: Implementar herramientas de automatización de pruebas como Selenium, TestNG o Cypress para automatizar las pruebas unitarias, de integración y de aceptación, asegurando la calidad del software y reduciendo los tiempos de prueba

- Escenario Nube: Utilizar contenedores Docker y plataformas de orquestación como Kubernetes o Amazon ECS (Elastic Container Service) para empaquetar y desplegar aplicaciones rápidamente, facilitando la gestión de dependencias y la escalabilidad

- Escenario Nube: Implementar herramientas de automatización de seguridad como AWS Security Hub, Google Cloud Security Command Center o Azure Security Center para monitorear y asegurar continuamente la infraestructura y las aplicaciones, garantizando el cumplimiento normativo.

- Escenario OnPremise: Utilizar herramientas de CI/CD como Jenkins, GitLab CI, Bamboo o CircleCI para automatizar la construcción, prueba y despliegue de aplicaciones en el entorno OnPremise, permitiendo iteraciones rápidas y despliegues frecuentes

- Escenario OnPremise: Utilizar herramientas de monitoreo y alertas automatizadas como Prometheus con Alertmanager, Nagios o Zabbix para detectar y responder a problemas en tiempo real, asegurando la disponibilidad y el rendimiento de las aplicaciones

16.4. Minimización de dependencias externas

Se considera para escenarios donde es necesario reducir la dependencia de componentes externos y terceros que puedan ralentizar el desarrollo o introducir incertidumbre en el proceso

Potenciales decisiones de arquitectura

- Optar por servicios nativos del proveedor de nube (como bases de datos, almacenamiento, y servicios de mensajería) en lugar de soluciones de terceros, para reducir las dependencias externas y aprovechar la integración y optimización nativa del proveedor.

- Utilizar plataformas de Functions-as-a-Service (FaaS) como AWS Lambda, Google Cloud Functions o Azure Functions para ejecutar código sin necesidad de gestionar servidores, reduciendo la complejidad y la dependencia de infraestructura externa

- Diseñar microservicios que sean lo más autónomos posible, minimizando las dependencias entre ellos y con servicios externos, permitiendo iteraciones y despliegues más rápidos

- Emplear librerías y SDKs oficiales proporcionados por el proveedor de nube, que suelen estar mejor soportados y actualizados, reduciendo la necesidad de integrar y mantener librerías de terceros.

- Utilizar estándares y protocolos abiertos (como REST, JSON, o gRPC) para la comunicación entre servicios, en lugar de depender de soluciones propietarias que puedan dificultar la interoperabilidad y la flexibilidad

- Utilizar herramientas nativas de gestión de dependencias y automatización como AWS CloudFormation, Google Deployment Manager o Azure Resource Manager, que están optimizadas para trabajar con los servicios de nube del proveedor

- Desarrollar y utilizar herramientas de automatización interna (como scripts y sistemas de gestión de configuración como Ansible, Puppet o Chef) para gestionar dependencias y despliegues sin necesidad de servicios externos

- Emplear contenedores Docker y plataformas de orquestación como Kubernetes, OpenShift o Docker Swarm en la infraestructura OnPremise para empaquetar y desplegar aplicaciones de manera consistente y eficiente, minimizando la dependencia de plataformas externas

- Desarrollar APIs y servicios internos que puedan ser reutilizados por diferentes equipos y aplicaciones dentro de la organización, reduciendo la necesidad de integrar soluciones externas

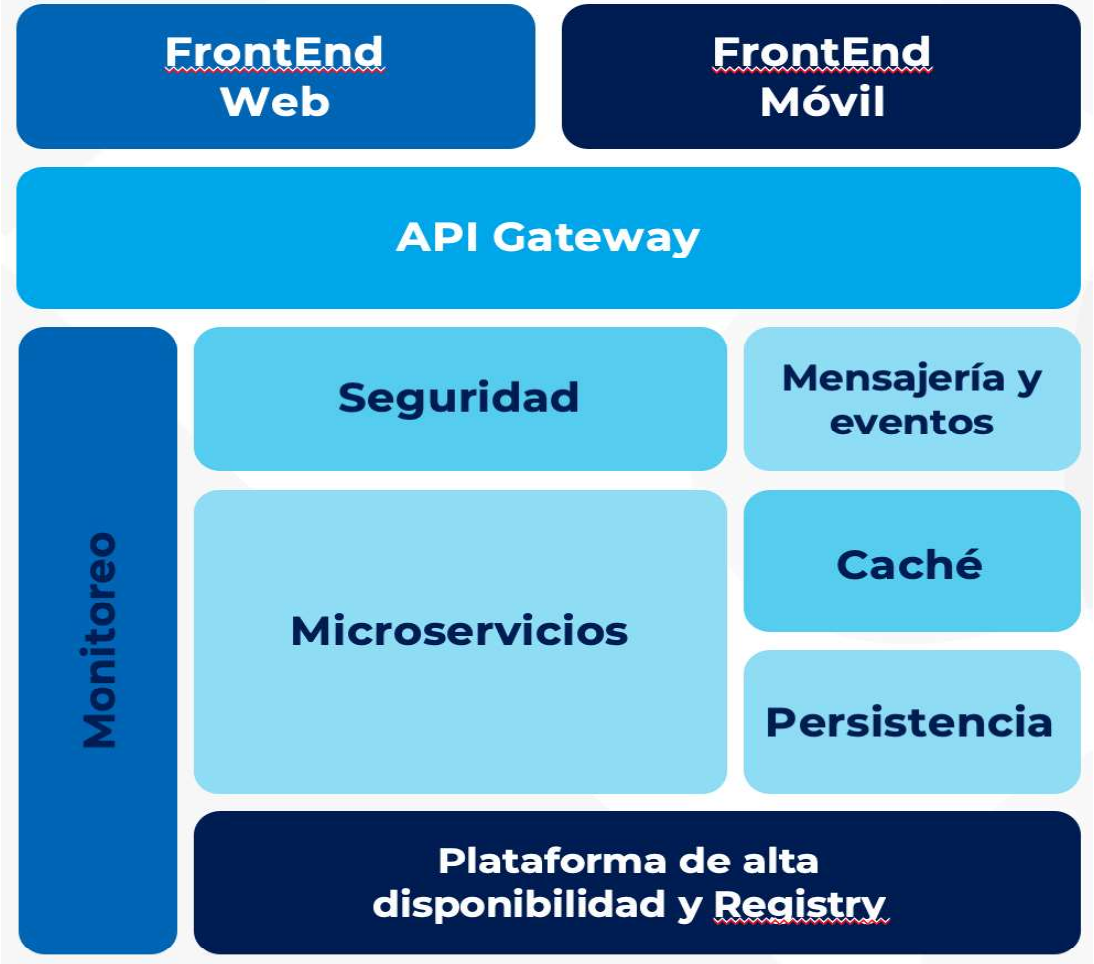
17. Gobierno

Situación problema:	¿Cómo estamos asegurando que nuestro sistema cumpla con regulaciones, estándares y políticas internas y externas?
Motivación:	El gobierno garantiza la coherencia, la conformidad y la calidad del sistema a lo largo de su ciclo de vida. Atender este atributo nos permite mitigar riesgos, optimizar recursos y mantener la confianza de las partes interesadas en el sistema.

El grupo de arquitectura de soluciones de catastro el cual pertenece a la alcaldía de Medellín deberá crear un grupo interdisciplinario que se encargue del gobierno de la plataforma.

18. Arquitectura de Referencia

La arquitectura de referencia propuesta está diseñada para optimizar la escalabilidad, la seguridad y la eficiencia operativa en un entorno distribuido tanto en Nube como en OnPremise. Basada en principios de arquitectura moderna y tecnologías adoptadas por la industria, la arquitectura se divide en siete componentes clave que trabajan en conjunto para soportar integraciones ágiles complejas y de alta disponibilidad. En la Figura se observan los diferentes componentes clave:



Componentes Arquitectura de Referencia

Componentes de la Arquitectura de referencia:

A continuación, se muestra un esquema de correspondencia, donde se mapea cada uno de los componentes clave de la arquitectura de referencia, con las potenciales decisiones arquitectónicas que se detallaron en el capítulo anterior.

Uno a uno, para cada componente clave se ampliarán las decisiones arquitectónicas, las tecnologías sugeridas y los patrones y principios en los que se basan. Tambien se puede observar en las tablas unas franjas de color verde que inidcan la viabilidad de estas herramientas para la plataforma de Catastro Virtual.

18.1. Componente clave: Monitoreo

Monitoreo	Decisiones arquitectónicas Registro y seguimiento de eventos Monitoreo de métricas de rendimiento Patrones Monitoreo basado en logs Consumer Adapter Event Message

18.2. Tecnologías sugeridas: Registro y seguimiento de eventos

Herramienta/ Tecnología	Descripción	Beneficios	Cloud/OnPremise
ELK Stack (Elasticsearch, Logstash, Kibana)	Solución popular para la búsqueda, análisis y visualización de logs en tiempo real.	Centralización de logs, análisis avanzado, visualización en tiempo real.	Ambos
Splunk	Plataforma para la búsqueda, monitoreo y análisis de datos generados por máquinas.	Escalabilidad, capacidades avanzadas de alertas y monitoreo, amplias integraciones.	Ambos
Graylog	Plataforma de gestión de logs de código abierto.	Interfaz intuitiva, extensibilidad, eficiencia en el procesamiento de logs.	Ambos
Fluentd	Colector de datos de logs de código abierto.	Flexibilidad, extensibilidad con más de 300 plugins, bajo consumo de recursos.	Ambos
Amazon CloudWatch Logs	Servicio para monitorear, almacenar y acceder a archivos de registro de sus recursos de AWS	Centralización de logs, integración nativa con otros servicios de AWS, alertas y monitoreo en tiempo real.	Cloud (AWS)
AWS CloudTrail	Servicio que proporciona registros de auditoría para su cuenta de AWS, incluyendo actividad del usuario, actividad de la API y más.	Registro detallado de todas las actividades en la cuenta de AWS, importante para la seguridad y auditoría.	Cloud (AWS)
Google Cloud Logging	Servicio que permite almacenar, buscar, analizar y alertar sobre datos de registro en tiempo real.	Integración con otros servicios de Google Cloud, análisis en tiempo real, alertas configurables.	Cloud (Google)
Google Cloud Audit Logs	Servicio que registra las actividades administrativas y de acceso a los datos en los servicios de Google Cloud.	Auditoría detallada, cumplimiento normativo, visibilidad de acceso y cambios en los recursos.	Cloud (Google)

Azure Monitor Logs (Log Analytics):	Plataforma de gestión de datos de registro y telemetría que permite recopilar, analizar y actuar sobre datos de registro.	Centralización de logs, análisis avanzado, integración con Azure Monitor para alertas y visualización.	Cloud (Azure)
Azure Activity Log:	Servicio que proporciona un registro de todas las actividades en su cuenta de Azure.	Registro detallado de actividades administrativas, importante para la seguridad y auditoría.	Cloud (Azure)

18.3. Tecnologías sugeridas: Monitoreo de métricas de rendimiento

Herramienta/ Tecnología	Descripción	Beneficios	Cloud/OnPremise
Prometheus	Herramienta de monitoreo y alerta de código abierto para recopilar y almacenar métricas en series temporales.	Potente motor de consulta (PromQL), integración con Alertmanager, integración con Grafana.	Ambos
Grafana	Plataforma de código abierto para analítica y monitoreo, compatible con múltiples fuentes de datos.	Visualización interactiva, compatibilidad con múltiples backends de datos, funcionalidades de alertas.	Ambos
Nagios	Herramienta de monitoreo de sistemas, redes e infraestructura.	Extensibilidad con plugins, potentes capacidades de alertas y notificaciones, escalabilidad.	OnPremise
Zabbix	Solución de monitoreo de código abierto para redes, servidores y aplicaciones.	Monitoreo integral, robusto sistema de alertas, adecuado para grandes entornos empresariales.	Ambos
Amazon CloudWatch	Servicio para monitorear y gestionar métricas, establecer alarmas y responder a cambios en su infraestructura de AWS.	Monitoreo en tiempo real, alertas configurables, integración nativa con otros servicios de AWS.	Cloud (AWS)
AWS X-Ray	Servicio que ayuda a los desarrolladores a analizar y depurar aplicaciones distribuidas en producción.	Trazabilidad distribuida, identificación de cuellos de botella y problemas de rendimiento.	Cloud (AWS)
Google Cloud Monitoring (anteriormente Stackdriver)	Servicio que proporciona monitoreo, alertas y visualización de métricas de rendimiento de sus recursos en Google Cloud.	Integración con otros servicios de Google Cloud, alertas configurables, dashboards personalizados.	Cloud (AWS)
Google Cloud Trace:	Servicio de trazabilidad distribuida que permite ver	Identificación de latencias y cuellos de botella,	Cloud (Google)

	y analizar el rendimiento de las solicitudes en su aplicación.	integración con otros servicios de Google Cloud.	
Azure Monitor:	Plataforma completa de monitoreo que proporciona métricas de rendimiento, alertas y análisis para recursos en Azure.	Integración con otros servicios de Azure, dashboards personalizados, alertas y notificaciones.	Cloud (Azure)
Azure Application Insights:	Servicio para el monitoreo de aplicaciones que proporciona métricas de rendimiento y trazabilidad de solicitudes.	Trazabilidad de extremo a extremo, identificación de problemas de rendimiento, integración con Azure DevOps.	Cloud (Azure)

18.4. Componente clave: Seguridad

Seguridad	Decisiones arquitectónicas Autenticación y autorización Cifrado Gestión de Identidades Políticas y estándares Gestión de cambios Auditoría y cumplimiento Autorización de APIs y servicios Acceso a servicios vía HTTP y HTTPS Patrones Twelve factor App Image-based integration deployment

18.5. Tecnologías sugeridas: Autenticación y Autorización

Herramienta/ Tecnología	Descripción	Beneficios	Cloud/OnPremise
AWS IAM	Servicio de AWS para gestionar identidades y accesos de usuarios y recursos.	Control centralizado, integración nativa con servicios de AWS, gestión granular de permisos.	Cloud (AWS)
Azure Active Directory (Azure AD)	Servicio de Azure para gestión de identidades y accesos, con soporte para SSO y MFA.	SSO, MFA, integración con aplicaciones de Azure y Microsoft 365, cumplimiento normativo.	Cloud (Azure)
Google Cloud Identity	Servicio de Google Cloud para gestionar identidades y accesos, con soporte para SSO y MFA.	SSO, MFA, integración con servicios de Google Cloud, control centralizado de accesos.	Cloud (Google Cloud)
Okta	Plataforma de identidad y gestión de accesos con soporte para SSO y MFA.	SSO, MFA, integración con múltiples aplicaciones, políticas de seguridad avanzadas.	Ambos

Auth0	Plataforma de autenticación y autorización con soporte para SSO y MFA.	Flexibilidad en la integración, fácil de implementar, soporte para múltiples métodos de autenticación.	Ambos
LDAP (Lightweight Directory Access Protocol)	Protocolo estándar para acceder y mantener servicios de directorio de información distribuida.	Gestión centralizada de usuarios, integración con múltiples aplicaciones, robusto y probado.	OnPremise
Active Directory (AD)	Servicio de directorio de Microsoft para gestión de identidades y accesos en entornos Windows.	Gestión centralizada, integración con aplicaciones de Windows, políticas de grupo y seguridad avanzadas.	OnPremise
Keycloak	Plataforma de código abierto para gestión de identidades y accesos con soporte para SSO y MFA.	SSO, MFA, integración con múltiples aplicaciones, altamente configurable.	Ambos
OneLogin	Plataforma de identidad y gestión de accesos con soporte para SSO y MFA.	SSO, MFA, integración con múltiples aplicaciones, interfaz intuitiva y fácil de usar.	Ambos
Ping Identity	Solución de gestión de identidades y accesos con soporte para SSO y MFA.	SSO, MFA, integración con aplicaciones empresariales, políticas de seguridad avanzadas.	Ambos

18.6. Tecnologías sugeridas: Cifrado

Herramienta/ Tecnología	Descripción	Beneficios	Cloud/OnPremise
AWS Key Management Service (KMS)	Servicio gestionado de AWS para crear y controlar claves de cifrado.	Gestión centralizada de claves, integración nativa con otros servicios de AWS, escalabilidad.	Cloud (AWS)
Azure Key Vault	Servicio de Azure para la gestión de claves, secretos y certificados.	Seguridad robusta, integración con servicios de Azure, simplifica la gestión de claves y secretos.	Cloud (Azure)
Google Cloud Key Management Service (KMS)	Servicio de Google Cloud para gestionar claves criptográficas.	Control centralizado, integración con otros servicios de Google Cloud, cumplimiento normativo.	Cloud (Google Cloud)
HashiCorp Vault	Herramienta de código abierto para gestionar secretos y proteger datos sensibles.	Alta flexibilidad, soporte para múltiples backends, políticas de acceso detalladas.	Ambos

OpenSSL	Biblioteca de software para aplicaciones seguras que permite el cifrado de datos.	Amplia compatibilidad, probado y robusto, soporte para múltiples algoritmos de cifrado.	Ambos
BitLocker	Funcionalidad de cifrado de disco completo integrada en Windows.	Protección de datos en reposo, fácil de usar, integración nativa con Windows.	OnPremise
LUKS (Linux Unified Key Setup)	Estándar de cifrado de disco en Linux.	Seguridad robusta, cifrado de disco completo, soporte nativo en muchas distribuciones Linux.	OnPremise
VeraCrypt	Software de cifrado de disco de código abierto.	Protección de datos en reposo, soporte multiplataforma, cifrado de particiones y discos completos.	Ambos
Thales CipherTrust	Plataforma de gestión de claves y cifrado empresarial.	Gestión centralizada, cumplimiento normativo, soporte para entornos híbridos.	Ambos
AWS CloudHSM	Servicio de AWS que proporciona módulos de seguridad de hardware (HSM) dedicados.	Protección de claves criptográficas, cumplimiento con normativas de alta seguridad, integración con AWS KMS.	Cloud (AWS)

18.7. Tecnologías sugeridas: Gestión de Identidades

Herramienta/ Tecnología	Descripción	Beneficios	Cloud/OnPremise
AWS IAM	Servicio gestionado por AWS para gestionar identidades y accesos.	Control centralizado de permisos, integración nativa con servicios de AWS, políticas de acceso detalladas.	Cloud (AWS)
Azure Active Directory (Azure AD)	Servicio de Microsoft para la gestión de identidades y accesos con soporte para SSO y MFA.	SSO, MFA, integración con Microsoft 365 y otros servicios de Azure, cumplimiento normativo.	Cloud (Azure)
Google Cloud Identity	Servicio de Google Cloud para gestionar identidades y accesos.	Control centralizado, SSO, MFA, integración con servicios de Google Cloud.	Cloud (Google Cloud)
Okta	Plataforma de identidad y gestión de accesos con soporte para SSO y MFA.	SSO, MFA, integración con múltiples aplicaciones, políticas de seguridad avanzadas.	Ambos

Auth0	Plataforma de autenticación y autorización con soporte para SSO y MFA.	Flexibilidad en la integración, fácil de implementar, soporte para múltiples métodos de autenticación.	Ambos
LDAP (Lightweight Directory Access Protocol)	Protocolo estándar para acceder y mantener servicios de directorio de información distribuida.	Gestión centralizada de usuarios, integración con múltiples aplicaciones, robusto y probado.	OnPremise
Active Directory (AD)	Servicio de directorio de Microsoft para la gestión de identidades y accesos en entornos Windows.	Gestión centralizada, integración con aplicaciones de Windows, políticas de grupo y seguridad avanzadas.	OnPremise
Keycloak	Plataforma de código abierto para la gestión de identidades y accesos con soporte para SSO y MFA.	SSO, MFA, integración con múltiples aplicaciones, altamente configurable.	Ambos
OneLogin	Plataforma de identidad y gestión de accesos con soporte para SSO y MFA.	SSO, MFA, integración con múltiples aplicaciones, interfaz intuitiva y fácil de usar.	Ambos
Ping Identity	Solución de gestión de identidades y accesos con soporte para SSO y MFA.	SSO, MFA, integración con aplicaciones empresariales, políticas de seguridad avanzadas.	Ambos

18.8. Tecnologías sugeridas: Políticas y estándares

Herramienta/Tecnología	Descripción	Beneficios	Cloud (Cuál)/OnPremise
AWS Organizations	Servicio de AWS para gestionar y aplicar políticas y estándares en múltiples cuentas de AWS.	Gestión centralizada de políticas, simplifica la gobernanza, integración nativa con servicios de AWS.	Cloud (AWS)
Azure Policy	Servicio de Azure para crear, asignar y gestionar políticas en los recursos de Azure.	Gestión centralizada, cumplimiento normativo, auditoría y remediación automática.	Cloud (Azure)
Google Cloud Organization Policy	Servicio de Google Cloud para definir y aplicar políticas a nivel de organización.	Control centralizado, cumplimiento normativo, integración con otros servicios de Google Cloud.	Cloud (Google Cloud)
HashiCorp Sentinel	Framework de políticas como código para gestionar políticas en infraestructura como código.	Políticas como código, integración con Terraform y otras herramientas de HashiCorp, altamente configurable.	Ambos

Open Policy Agent (OPA)	Motor de políticas de código abierto que permite definir políticas como código para aplicaciones y microservicios.	Políticas como código, integración con Kubernetes y otras plataformas, flexible y extensible.	Ambos
CIS-CAT Pro	Herramienta de configuración de auditoría y evaluación basada en estándares CIS.	Auditorías automatizadas, cumplimiento con estándares CIS, informes detallados de cumplimiento.	Ambos
AWS Config	Servicio de AWS para evaluar, auditar y monitorear configuraciones de recursos de AWS.	Monitoreo continuo, cumplimiento normativo, integración nativa con servicios de AWS.	Cloud (AWS)
Microsoft Defender for Cloud	Servicio de Azure para evaluar y aplicar estándares de seguridad en entornos de Azure y híbridos.	Evaluaciones de seguridad continuas, cumplimiento normativo, integración con otros servicios de Azure.	Cloud (Azure)
Chef InSpec	Herramienta de código abierto para pruebas de cumplimiento automatizadas y auditorías.	Pruebas de cumplimiento automatizadas, integración con Chef y otras herramientas de DevOps, flexible y extensible.	Ambos
Puppet Remediate	Herramienta para la remediación automatizada de problemas de cumplimiento en infraestructura.	Remediación automatizada, integración con Puppet y otras herramientas de DevOps, informes de cumplimiento.	Ambos

18.9. Tecnologías sugeridas: Gestión de Cambio

Herramienta/Tecnología	Descripción	Beneficios	Cloud (Cuál)/OnPremise
ServiceNow Change Management	Plataforma de gestión de servicios que incluye módulos para la gestión de cambios.	Gestión centralizada de cambios, flujos de trabajo personalizados, integración con ITSM.	Ambos
Jira Service Management	Herramienta de Atlassian para la gestión de servicios y cambios, con soporte para ITIL.	Gestión de cambios basada en ITIL, integración con Jira, flujos de trabajo personalizables.	Ambos
BMC Remedy Change Management	Solución de gestión de servicios empresariales que incluye módulos para la gestión de cambios.	Gestión de cambios completa, cumplimiento con ITIL, integración con otros productos de BMC.	Ambos
GitLab	Plataforma DevOps que incluye gestión de cambios a través de CI/CD y merge requests.	Gestión integrada de cambios y código, automatización de CI/CD, visibilidad del ciclo de vida del desarrollo.	Ambos

GitHub Actions	Funcionalidad de GitHub para la automatización de flujos de trabajo de CI/CD.	Automatización de cambios a través de CI/CD, integración con repositorios de código, visibilidad y control.	Ambos
Puppet Enterprise	Plataforma de automatización que incluye capacidades de gestión de cambios en la infraestructura.	Automatización de la gestión de cambios, visibilidad y control, informes detallados.	Ambos
Ansible Tower	Herramienta de Red Hat para la automatización y gestión de cambios en la infraestructura.	Automate changes, centralized management, integration with other Red Hat products.	Ambos
Chef Automate	Plataforma de gestión de cambios para infraestructura como código.	Gestión de cambios automatizada, integración con Chef, informes de cumplimiento.	Ambos
CA Service Desk Manager	Solución de gestión de servicios de CA Technologies con capacidades de gestión de cambios.	Gestión centralizada de cambios, flujos de trabajo personalizados, cumplimiento con ITIL.	Ambos
Azure DevOps	Plataforma de Microsoft para DevOps que incluye gestión de cambios a través de pipelines de CI/CD.	Gestión integrada de cambios y despliegues, visibilidad del ciclo de vida del desarrollo, integración con Azure.	Cloud (Azure)

18.10. Tecnologías sugeridas: Políticas y estándares

Herramienta/ Tecnología	Descripción	Beneficios	Cloud/OnPremise
AWS Organizations	Servicio de AWS para gestionar políticas y estándares en múltiples cuentas.	Gestión centralizada, simplificación de la gobernanza, integración nativa con servicios de AWS.	Cloud (AWS)
Azure Policy	Servicio de Azure para crear, asignar y gestionar políticas en los recursos de Azure.	Cumplimiento normativo, auditoría y remediación automática, gestión centralizada.	Cloud (Azure)
Google Cloud Organization Policy	Servicio de Google Cloud para definir y aplicar políticas a nivel de organización.	Control centralizado, integración con otros servicios de Google Cloud, cumplimiento normativo.	Cloud (Google Cloud)
HashiCorp Sentinel	Framework de políticas como código para gestionar políticas en infraestructura como código.	Políticas como código, integración con Terraform y otras herramientas de HashiCorp, configurable.	Ambos

Open Policy Agent (OPA)	Motor de políticas de código abierto para definir políticas como código para aplicaciones y microservicios.	Políticas como código, integración con Kubernetes y otras plataformas, flexible y extensible.	Ambos
CIS-CAT Pro	Herramienta de auditoría y evaluación basada en estándares CIS.	Auditorías automatizadas, cumplimiento con estándares CIS, informes detallados.	Ambos
AWS Config	Servicio de AWS para evaluar, auditar y monitorear configuraciones de recursos de AWS.	Monitoreo continuo, integración nativa con servicios de AWS, cumplimiento normativo.	Cloud (AWS)
Microsoft Defender for Cloud	Servicio de Azure para evaluar y aplicar estándares de seguridad en entornos de Azure e híbridos.	Evaluaciones de seguridad continuas, cumplimiento normativo, integración con servicios de Azure.	Cloud (Azure)
Chef InSpec	Herramienta de código abierto para pruebas de cumplimiento automatizadas y auditorías.	Pruebas de cumplimiento automatizadas, integración con Chef y otras herramientas de DevOps.	Ambos
Puppet Remediate	Herramienta para la remediación automatizada de problemas de cumplimiento en infraestructura.	Remediación automatizada, integración con Puppet, informes de cumplimiento.	Ambos

18.11. Tecnologías sugeridas: Gestión de Cambios

Herramienta/Tecnología	Descripción	Beneficios	Cloud (Cuál)/OnPremise
ServiceNow Change Management	Plataforma de gestión de servicios que incluye módulos para la gestión de cambios.	Gestión centralizada de cambios, flujos de trabajo personalizados, integración con ITSM.	Ambos
Jira Service Management	Herramienta de Atlassian para la gestión de servicios y cambios, con soporte para ITIL.	Gestión de cambios basada en ITIL, integración con Jira, flujos de trabajo personalizables.	Ambos
Jenkins	Jenkins es una herramienta de automatización de código abierto diseñada para facilitar la integración continua y la entrega continua (CI/CD).	Facilita la integración continua, permitiendo detectar y resolver problemas de manera temprana, mejorando la calidad del código.	Ambos
BMC Remedy Change Management	Solución de gestión de servicios empresariales que incluye módulos para la gestión de cambios.	Gestión de cambios completa, cumplimiento con ITIL, integración con otros productos de BMC.	Ambos
GitLab	Plataforma DevOps que incluye gestión de cambios a través de CI/CD y merge requests.	Gestión integrada de cambios y código, automatización de CI/CD, visibilidad del ciclo de vida del desarrollo.	Ambos
GitHub Actions	Funcionalidad de GitHub para la automatización de flujos de trabajo de CI/CD.	Automatización de cambios a través de CI/CD, integración con repositorios de código, visibilidad y control.	Ambos
Puppet Enterprise	Plataforma de automatización que incluye capacidades de gestión de cambios en la infraestructura.	Automatización de la gestión de cambios, visibilidad y control, informes detallados.	Ambos

18.12. Componente clave: Integración o Mediación

Integración o mediación	Decisiones arquitectónicas Disponibilidad Desempeño Interoperabilidad Mantenibilidad Time-To-Market Patrones API File Transfer Messaging

18.13. Tecnologías sugeridas: Disponibilidad

Herramienta/Tecnología	Descripción	Beneficios	Cloud (Cuál)/OnPremise
Kubernetes	Plataforma de orquestación de contenedores que facilita la implementación y gestión de aplicaciones altamente disponibles.	Escalabilidad, tolerancia a fallos, gestión automatizada de recursos, integración con servicios de nube.	Ambos
Docker Swarm	Herramienta de orquestación de contenedores que permite la alta disponibilidad de aplicaciones distribuidas en clústeres.	Facilidad de uso, escalabilidad, tolerancia a fallos, integración con Docker y otras herramientas.	Ambos
Apache Kafka	Plataforma de mensajería distribuida diseñada para la alta disponibilidad y tolerancia a fallos.	Escalabilidad horizontal, tolerancia a fallos, replicación de datos, integración con sistemas heterogéneos.	Ambos
Redis	Base de datos en memoria que ofrece alta disponibilidad a través de la replicación y el clustering.	Tiempos de respuesta ultrarrápidos, replicación de datos, particionamiento, soporte para almacenamiento en caché.	Ambos
ZooKeeper	Servicio de coordinación distribuida utilizado para la gestión de configuraciones y el mantenimiento de la alta disponibilidad.	Coordinación distribuida, elección del líder, gestión de configuraciones, integración con sistemas distribuidos.	Ambos
Consul	Herramienta de descubrimiento de servicios y gestión de configuraciones que garantiza la alta disponibilidad de los servicios.	Descubrimiento de servicios, monitorización de la salud, gestión de configuraciones, tolerancia a fallos.	Ambos
Nginx	Servidor web y proxy inverso con capacidades de balanceo de carga y alta disponibilidad.	Balanceo de carga, tolerancia a fallos, escalabilidad, gestión de tráfico, soporte para aplicaciones web.	Ambos

HAProxy	Software de balanceo de carga y proxy TCP/HTTP que proporciona alta disponibilidad y escalabilidad.	Balanceo de carga, tolerancia a fallos, escalabilidad, soporte para SSL/TLS, configuración flexible.	Ambos
Amazon Route 53	Servicio de Amazon Web Services (AWS) que ofrece resolución de nombres de dominio (DNS) y gestión del tráfico.	Alta disponibilidad del DNS, enrutamiento de tráfico basado en políticas, monitoreo de la salud del endpoint.	Cloud (AWS)
Azure Traffic Manager	Servicio de Azure que proporciona enrutamiento de tráfico basado en la disponibilidad y el rendimiento de los endpoints.	Alta disponibilidad del DNS, enrutamiento de tráfico basado en políticas, monitoreo de la salud del endpoint.	Cloud (Azure)
Google Cloud Load Balancing	Servicio de Google Cloud que ofrece balanceo de carga global para aplicaciones altamente disponibles.	Balanceo de carga global, escalabilidad automática, distribución geográfica, integración con servicios de Google Cloud.	Cloud (Google Cloud)
NGINX Plus	Versión comercial de Nginx con características adicionales como balanceo de carga avanzado y monitoreo en tiempo real.	Balanceo de carga avanzado, monitoreo en tiempo real, soporte 24/7, integración con infraestructura existente.	Ambos

18.14. Tecnologías sugeridas: Desempeño

Herramienta/Tecnología	Descripción	Beneficios	Cloud (Cuál)/OnPremise
Apache Kafka	Plataforma de mensajería distribuida diseñada para manejar flujos de datos en tiempo real.	Alta escalabilidad, bajo retardo, procesamiento distribuido, tolerancia a fallos, integración con sistemas heterogéneos.	Ambos
RabbitMQ	Sistema de mensajería de código abierto que implementa el protocolo AMQP para la comunicación entre aplicaciones.	Alta disponibilidad, baja latencia, soporte para múltiples protocolos, encolamiento de mensajes, integración con sistemas de middleware.	Ambos
SQS (Amazon)			Cloud (AWS)
ActiveMQ			Ambos
Redis	Base de datos en memoria de código abierto que se utiliza como caché, base de datos y agente de mensajería.	Alta velocidad, baja latencia, almacenamiento en caché de datos, estructuras de datos flexibles, alta disponibilidad.	Ambos
Apache Flink	Motor de procesamiento de datos distribuido y	Procesamiento de datos ultrarrápido, baja latencia,	Ambos

	tolerante a fallos para realizar análisis en tiempo real y procesamiento de datos por lotes.	tolerancia a fallos, integración con Apache Kafka, soporte para múltiples fuentes de datos.	
Apache Spark	Motor de análisis de datos distribuido diseñado para el procesamiento de grandes volúmenes de datos de forma rápida y eficiente.	Procesamiento de datos en memoria, tolerancia a fallos, integración con Apache Kafka, soporte para múltiples fuentes de datos, fácil de usar.	Ambos
NGINX	Servidor web y proxy inverso con capacidades de balanceo de carga y almacenamiento en caché.	Balanceo de carga, almacenamiento en caché, baja latencia, alta disponibilidad, escalabilidad horizontal y vertical.	Ambos
HAProxy	Software de balanceo de carga y proxy TCP/HTTP que proporciona alta disponibilidad y escalabilidad.	Balanceo de carga, alta disponibilidad, baja latencia, escalabilidad horizontal y vertical, configuración flexible.	Ambos
Envoy Proxy	Proxy de código abierto diseñado para la comunicación entre microservicios y aplicaciones distribuidas.	Balanceo de carga, enrutamiento dinámico, seguridad, observabilidad, bajo retardo, integración con Kubernetes y otros sistemas de orquestación.	Ambos
Varnish Cache	Servidor proxy de alta velocidad y código abierto diseñado para mejorar el rendimiento de sitios web al almacenar en caché las solicitudes frecuentes.	Almacenamiento en caché, aceleración web, balanceo de carga, escalabilidad, alta disponibilidad, seguridad.	Ambos
Amazon ElastiCache	Servicio de caché completamente administrado y compatible con Redis o Memcached, proporcionado por AWS.	Escalabilidad, alta disponibilidad, seguridad, monitorización, integración con servicios de AWS, gestión automatizada.	Cloud (AWS)
Azure Cache for Redis	Servicio de caché completamente administrado y compatible con Redis, proporcionado por Microsoft Azure.	Escalabilidad, alta disponibilidad, seguridad, monitorización, integración con servicios de Azure, gestión automatizada.	Cloud (Azure)

Google Cloud Memorystore	Servicio de caché completamente administrado y compatible con Redis, proporcionado por Google Cloud Platform.	Escalabilidad, alta disponibilidad, seguridad, monitorización, integración con servicios de Google Cloud, gestión automatizada.	Cloud (Google Cloud)

18.15. Tecnologías sugeridas: Interoperabilidad

Apache Kafka	Plataforma de mensajería distribuida diseñada para manejar flujos de datos en tiempo real.	Alta escalabilidad, soporte para múltiples protocolos, procesamiento distribuido, baja latencia, tolerancia a fallos, integración con sistemas heterogéneos.	Ambos
Apache Camel	Framework de integración de código abierto que proporciona implementaciones de patrones de integración empresarial a través de una variedad de protocolos y tecnologías de transporte.	Facilita la integración entre diferentes sistemas y aplicaciones, implementación de patrones de integración comunes, soporte para un amplio rango de protocolos y formatos de datos, modularidad y extensibilidad, amplia comunidad y soporte.	Ambos
MuleSoft Anypoint Platform	Plataforma de integración empresarial que facilita la conectividad de sistemas y aplicaciones a través de API, conectores y herramientas de desarrollo.	Facilita la creación, publicación, gestión y supervisión de API, integración con sistemas existentes y servicios en la nube, automatización de procesos, orquestación de servicios, seguridad de extremo a extremo, análisis y monitorización de la actividad.	Cloud (MuleSoft)
Apache CXF	Framework de servicios web de código abierto que permite desarrollar y desplegar servicios web utilizando una variedad de estándares y protocolos, como SOAP y REST.	Implementación de servicios web interoperables, soporte para diferentes protocolos y formatos de datos, integración con tecnologías Java EE y Spring, seguridad y	Ambos

		políticas de servicios, amplia comunidad de usuarios y soporte.	
WSO2 Enterprise Integrator	Plataforma de integración de código abierto que ofrece capacidades de integración, servicios y procesos empresariales, incluyendo ESB, servicios de API, integración de datos y orquestación de servicios.	Facilita la integración de sistemas y aplicaciones heterogéneas, implementación y gestión de servicios de API, orquestación de procesos empresariales, seguridad y gestión de identidades, monitorización y análisis de la actividad.	Ambos
Talend Integration Suite	Suite de integración de datos de código abierto que permite la integración de datos en tiempo real y por lotes, migración de datos, calidad de datos, gestión de API y procesamiento de eventos.	Facilita la integración de datos entre sistemas y aplicaciones, soporte para diferentes fuentes y destinos de datos, diseño visual de procesos de integración, automatización de tareas, gestión y monitorización centralizada, comunidad y soporte activos.	Ambos
IBM Integration Bus	Plataforma de integración empresarial que proporciona capacidades de mensajería, enrutamiento, transformación de datos y adaptación a través de una amplia gama de protocolos y estándares de la industria.	Facilita la integración de aplicaciones y sistemas heterogéneos, soporte para protocolos y formatos de datos estándar, orquestación de servicios, monitorización y gestión centralizada, seguridad y gestión de identidades, escalabilidad y disponibilidad.	Ambos
Red Hat Fuse	Plataforma de integración de código abierto basada en Apache Camel y fabric8 que proporciona capacidades de mensajería, transformación, enrutamiento y orquestación de servicios para la integración de aplicaciones empresariales.	Facilita la implementación de patrones de integración empresarial, conectividad de aplicaciones y sistemas, gestión de API y microservicios, despliegue en contenedores, monitorización y gestión centralizada, seguridad y gobierno de servicios.	Ambos

WSO2 API Manager	Plataforma de gestión de API de código abierto que facilita la creación, publicación, supervisión y monetización de API, así como la gestión de desarrolladores y la aplicación de políticas de seguridad y acceso.	Facilita la creación y gestión de API, publicación y descubrimiento de servicios, seguridad de extremo a extremo, control de acceso y autenticación, monitorización y análisis de uso, gestión de versiones y ciclos de vida.	Ambos
AWS Lambda	Servicio de cómputo sin servidor de AWS que permite ejecutar código en respuesta a eventos sin tener que aprovisionar o administrar servidores.	Escalabilidad automática, pago por uso, integración con servicios de AWS y eventos, ejecución de código personalizado, gestión y monitorización centralizada, seguridad y control de acceso.	Cloud (AWS)
Azure Logic Apps	Servicio de automatización de flujo de trabajo de Microsoft Azure que permite orquestar y automatizar tareas y procesos empresariales a través de una amplia gama de servicios y aplicaciones en la nube y locales.	Diseño visual de flujos de trabajo, conectividad con más de 200 servicios y aplicaciones, integración con servicios de Azure y eventos, monitorización y gestión centralizada, seguridad y cumplimiento de normativas.	Cloud (Azure)
Google Cloud Functions	Servicio de cómputo sin servidor de Google Cloud Platform que permite ejecutar código en respuesta a eventos sin necesidad de aprovisionar o administrar servidores.	Escalabilidad automática, pago por uso, integración con servicios de Google Cloud y eventos, ejecución de código personalizado, gestión y monitorización centralizada, seguridad y control de acceso.	Cloud (Google Cloud)

18.16. Tecnologías sugeridas: Mantenibilidad

Herramienta/Tecnología	Descripción	Beneficios	Cloud (Cuál)/OnPremise
Apache Kafka	Plataforma de mensajería distribuida diseñada para manejar flujos de datos en tiempo real.	Alta escalabilidad, soporte para múltiples protocolos, procesamiento distribuido, baja latencia, tolerancia a fallos, integración con sistemas heterogéneos.	Ambos

Apache Camel	Framework de integración de código abierto que proporciona implementaciones de patrones de integración empresarial a través de una variedad de protocolos y tecnologías de transporte.	Facilita la integración entre diferentes sistemas y aplicaciones, implementación de patrones de integración comunes, soporte para un amplio rango de protocolos y formatos de datos, modularidad y extensibilidad, amplia comunidad y soporte.	Ambos
MuleSoft Anypoint Platform	Plataforma de integración empresarial que facilita la conectividad de sistemas y aplicaciones a través de API, conectores y herramientas de desarrollo.	Facilita la creación, publicación, gestión y supervisión de API, integración con sistemas existentes y servicios en la nube, automatización de procesos, orquestación de servicios, seguridad de extremo a extremo, análisis y monitorización de la actividad.	Cloud (MuleSoft)
WSO2 Enterprise Integrator	Plataforma de integración de código abierto que ofrece capacidades de integración, servicios y procesos empresariales, incluyendo ESB, servicios de API, integración de datos y orquestación de servicios.	Facilita la integración de sistemas y aplicaciones heterogéneas, implementación y gestión de servicios de API, orquestación de procesos empresariales, seguridad y gestión de identidades, monitorización y análisis de la actividad.	Ambos
AWS Step Functions	Servicio de AWS que permite coordinar fácilmente los componentes de las aplicaciones distribuidas utilizando flujos de trabajo visuales basados en estados.	Facilita la orquestación de componentes de aplicaciones distribuidas, diseño visual de flujos de trabajo, ejecución confiable de tareas, integración con servicios de AWS, monitorización y gestión centralizadas, seguridad y control de acceso.	Cloud (AWS)
Azure Logic Apps	Servicio de automatización de flujo de trabajo de Microsoft Azure que permite orquestar y automatizar tareas y procesos empresariales a través de una amplia gama de servicios y aplicaciones en la nube y locales.	Diseño visual de flujos de trabajo, conectividad con más de 200 servicios y aplicaciones, integración con servicios de Azure y eventos, monitorización y gestión centralizada, seguridad y cumplimiento de normativas.	Cloud (Azure)
Google Cloud Composer	Servicio de Google Cloud que permite crear, programar y monitorizar flujos de trabajo basados en Apache Airflow para orquestar tareas en la	Facilita la creación y programación de flujos de trabajo, integración con servicios de Google Cloud y Apache Airflow, automatización y gestión	Cloud (Google Cloud)

	nube y en las instalaciones.	centralizada, seguridad y control de acceso.	

18.17. Tecnologías sugeridas: Time-To-Market

Herramienta/Tecnología	Descripción	Beneficios	Cloud (Cuál)/OnPremise
MuleSoft Anypoint Platform	Plataforma de integración empresarial que facilita la conectividad de sistemas y aplicaciones a través de API, conectores y herramientas de desarrollo.	Facilita el diseño y la implementación rápida de API y conectores, integración con sistemas existentes y servicios en la nube, automatización de procesos, orquestación de servicios, seguridad de extremo a extremo, análisis y monitorización de la actividad.	Cloud (MuleSoft)
Boomi Integration Platform	Plataforma de integración y gestión de datos basada en la nube que permite conectar aplicaciones, datos y dispositivos de forma rápida y eficiente.	Facilita la creación rápida de flujos de integración, conectividad con aplicaciones en la nube y locales, automatización de procesos, orquestación de servicios, transformación de datos, sincronización de datos, monitorización y gestión centralizada.	Cloud (Boomi)
AWS App Integration	Conjunto de servicios de AWS que ofrece diversas herramientas para facilitar la integración de aplicaciones y servicios en la nube. Esto incluye AWS Lambda, Amazon API Gateway, AWS Step Functions, Amazon EventBridge, Amazon SQS, Amazon SNS y otros servicios.	Facilita el desarrollo y despliegue rápido de aplicaciones, integración con servicios y eventos de AWS, implementación de arquitecturas sin servidor, gestión de API y eventos, orquestación de flujos de trabajo, mensajería y notificaciones, monitoreo y análisis avanzados.	Cloud (AWS)
Azure Integration Services	Conjunto de servicios de integración de Microsoft Azure que proporciona capacidades para conectar aplicaciones, datos y procesos empresariales en la nube y en las instalaciones. Incluye Azure Logic Apps, Azure API Management, Azure Service Bus, Azure Event Grid y otras herramientas.	Facilita la integración y orquestación de aplicaciones y servicios en Azure, conectividad con sistemas locales y en la nube, gestión de API, eventos y mensajería, automatización de flujos de trabajo, monitoreo y análisis avanzados, seguridad y cumplimiento de normativas.	Cloud (Azure)

Google Cloud Pub/Sub	Servicio de mensajería y publicación/suscripción de Google Cloud que permite la entrega de mensajes de forma asincrónica entre componentes de aplicaciones distribuidas.	Facilita la implementación de arquitecturas de microservicios y eventos, entrega de mensajes en tiempo real, escalabilidad automática, integración con servicios de Google Cloud, monitoreo y gestión centralizada, seguridad y cumplimiento de normativas.	Cloud (Google Cloud)
IBM Cloud Integration	Conjunto de servicios de integración de IBM Cloud que ofrece herramientas para conectar aplicaciones, datos y servicios en entornos de nube híbrida. Incluye IBM App Connect, IBM DataPower Gateway, IBM MQ, IBM API Connect y otras soluciones.	Facilita la integración de aplicaciones y datos en entornos de nube híbrida, conectividad segura con sistemas locales y en la nube, gestión de API, eventos y mensajería, transformación y enrutamiento de datos, monitoreo y análisis avanzados, seguridad y gobierno de servicios, cumplimiento de normativas.	Cloud (IBM Cloud)

18.18. Tecnologías sugeridas: File Transfer

Herramienta/Tecnología	Descripción	Beneficios	Cloud (Cuál)/OnPremise
IBM Aspera	IBM Aspera es una solución de transferencia de archivos de alta velocidad diseñada para mover grandes volúmenes de datos a través de redes globales, independientemente de la distancia o las condiciones de red. Utiliza tecnología patentada de transferencia de datos que optimiza la velocidad y la eficiencia, lo que permite transferencias rápidas y seguras incluso en redes de baja calidad.	Transferencia de archivos ultra rápida, seguridad avanzada con cifrado de extremo a extremo, gestión y monitoreo centralizados, soporte para transferencia de archivos de gran tamaño, integración con sistemas de almacenamiento en la nube y servidores locales, escalabilidad para gestionar volúmenes de datos masivos.	Ambos

SFTPPlus	SFTPPlus es un servidor de transferencia de archivos seguro que permite intercambiar datos de forma segura a través de redes, utilizando el protocolo SFTP (SSH File Transfer Protocol) y otros protocolos de transferencia de archivos. Proporciona características avanzadas de seguridad, como autenticación de dos factores, cifrado de extremo a extremo y políticas de acceso granular, junto con funciones de automatización y monitoreo.	Seguridad avanzada con cifrado y autenticación robustos, compatibilidad con múltiples protocolos de transferencia de archivos, capacidad de automatización y programación de transferencias, monitoreo en tiempo real de transferencias y actividad del servidor, integración con sistemas de autenticación existentes, opciones flexibles de implementación en la nube o en las instalaciones.	Ambos
Axway SecureTransport	Axway SecureTransport es una solución de transferencia de archivos gestionada que ofrece una plataforma segura y centralizada para el intercambio de archivos en empresas y organizaciones. Proporciona transferencias de archivos seguras y confiables, con capacidades avanzadas de seguridad, gestión de flujos de trabajo, monitoreo en tiempo real y cumplimiento de normativas, ayudando a proteger los datos confidenciales y a mantener la integridad de las transferencias.	Transferencias de archivos seguras y confiables, cumplimiento de normativas y estándares de seguridad, gestión centralizada de flujos de trabajo, monitoreo en tiempo real de la actividad del servidor, integración con sistemas existentes a través de API y herramientas de integración, opciones flexibles de implementación tanto en la nube como en las instalaciones.	Ambos

CrushFTP	CrushFTP es un servidor de transferencia de archivos que ofrece una solución segura y fácil de usar para compartir archivos a través de redes locales e Internet. Ofrece soporte para una amplia gama de protocolos de transferencia de archivos, incluyendo FTP, SFTP, FTPS, HTTP, HTTPS y WebDAV, junto con características de seguridad avanzadas como cifrado SSL/TLS, autenticación basada en certificados y filtros de IP.	Facilidad de uso con una interfaz intuitiva y basada en web, seguridad avanzada con cifrado SSL/TLS y autenticación robusta, soporte para múltiples protocolos de transferencia de archivos, características de automatización y programación, integración con sistemas de autenticación existentes, opciones flexibles de implementación en la nube o en las instalaciones.	Ambos
AWS Transfer for SFTP	AWS Transfer for SFTP es un servicio totalmente gestionado que permite transferir archivos de manera segura sobre el protocolo SFTP (SSH File Transfer Protocol) directamente en la nube de AWS. Elimina la necesidad de administrar servidores de SFTP y proporciona integración con otros servicios de AWS, como Amazon S3 y AWS Identity and Access Management (IAM), para simplificar y automatizar las operaciones de transferencia de archivos.	Gestión totalmente gestionada, escalabilidad automática según la demanda, integración nativa con otros servicios de AWS, seguridad avanzada con cifrado y autenticación, monitoreo y registro integrados, acceso a través de una interfaz de usuario basada en la web o API, reducción de costos al eliminar la necesidad de mantener servidores de SFTP locales.	Nube (AWS)

18.19. Componente clave: Mensajería y Eventos

Mensajería y eventos	Decisiones arquitectónicas Balanceo de carga Escalabilidad vertical y horizontal Protocolos de comunicación Patrones Messaging

18.20. Tecnologías sugeridas: Balanceo de Cargas

Herramienta/Tecnología	Descripción	Beneficios	Cloud (Cuál)/OnPremise

Kong APi Gateway	Kong API Gateway es una plataforma moderna y escalable para la gestión de APIs que actúa como intermediario entre los clientes (frontend web y móvil) y los microservicios backend. Su arquitectura basada en microservicios permite enrutar, autenticar, y proteger las solicitudes API, al tiempo que ofrece capacidades avanzadas como monitoreo, transformación de datos y control de tráfico. Kong es especialmente adecuado para entornos distribuidos y aplicaciones nativas en la nube, como la plataforma Catastro Virtual.	Gestión Centralizada de APIs: Simplifica la administración de todas las APIs mediante una única capa de control, reduciendo la complejidad operativa. Seguridad: Ofrece autenticación y autorización mediante OAuth2, OpenID Connect y otros estándares, garantizando el acceso seguro a los servicios. Escalabilidad: Diseñado para manejar grandes volúmenes de tráfico sin comprometer el rendimiento, lo que lo hace ideal para sistemas con alta concurrencia. Transformación de Datos: Permite adaptar las solicitudes y respuestas para cumplir con los formatos y requisitos del cliente o backend, minimizando esfuerzos en los servicios. Control de Tráfico: Implementa políticas de limitación de tasa (rate limiting), protegiendo los microservicios de sobrecargas. Monitoreo y Observabilidad: Proporciona métricas detalladas sobre el uso de APIs, integrándose con herramientas como Prometheus y Grafana para visualización y alertas. Extensibilidad: Su arquitectura basada en plugins permite añadir funcionalidades personalizadas según las necesidades del sistema. Compatibilidad: Soporta múltiples protocolos como REST, gRPC y GraphQL, garantizando la interoperabilidad con diversos servicios.	Ambos

NGINX	Servidor proxy de alto rendimiento y equilibrador de carga que puede utilizarse para distribuir el tráfico de manera uniforme entre múltiples servidores.	Alta escalabilidad, rendimiento y fiabilidad, configuración flexible, soporte para múltiples algoritmos de balanceo de carga, distribución de carga basada en la capacidad de los servidores, control de tráfico HTTP y TCP, monitorización y gestión centralizada.	Ambos
HAProxy	Software de equilibrador de carga y proxy TCP/HTTP de código abierto que ofrece balanceo de carga de nivel 4 y nivel 7, así como opciones de proxy inverso y detección de fallos de servidor.	Rendimiento excepcional, configuración flexible mediante reglas, distribución de carga basada en diferentes métricas (ponderación, IP-hash, etc.), soporte para múltiples protocolos y algoritmos de balanceo de carga, monitorización en tiempo real, alta disponibilidad y tolerancia a fallos.	Ambos
Apache Web Server	Servidor Web que se puede configurar como proxy equilibrador de carga que puede utilizarse para distribuir el tráfico de manera uniforme entre múltiples servidores.	Alta escalabilidad, rendimiento y fiabilidad, configuración flexible, soporte para múltiples algoritmos de balanceo de carga, distribución de carga basada en la capacidad de los servidores, control de tráfico HTTP y TCP, monitorización y gestión centralizada.	Ambos
Amazon Elastic Load Balancing (ELB)	Servicio de equilibrador de carga gestionado por AWS que distribuye automáticamente el tráfico entrante a través de varias instancias de Amazon EC2 o recursos de destino, como contenedores, direcciones IP y servicios de Lambda.	Configuración automática y escalado dinámico, distribución de carga basada en el tráfico y la salud de los recursos, monitoreo y ajuste automáticos, integración con servicios de AWS, soporte para varios tipos de balanceadores de carga (clásico, de aplicación y de red), seguridad y protección contra ataques DDoS, alta disponibilidad y fiabilidad.	Cloud (AWS)

Azure Load Balancer	Servicio de equilibrador de carga gestionado por Azure que distribuye el tráfico de red entrante entre las máquinas virtuales (VM) de Azure dentro de la misma región y virtual network.	Distribución de carga basada en reglas y algoritmos de equilibrio de carga, escalabilidad automática, integración con servicios de Azure, protección contra ataques DDoS, monitorización y alertas, alta disponibilidad y fiabilidad.	Cloud (Azure)
Google Cloud Load Balancing	Servicio de equilibrador de carga totalmente gestionado por Google Cloud que distribuye el tráfico entre múltiples instancias de VM, contenedores y servicios de Compute Engine o Kubernetes Engine.	Distribución de carga global y regional, escalado automático, configuración flexible, integración con servicios de Google Cloud, seguridad y protección contra ataques DDoS, monitorización y alertas, alta disponibilidad y fiabilidad.	Cloud (Google Cloud)
F5 BIG-IP	Plataforma de entrega de aplicaciones que incluye hardware y software de equilibrador de carga, proxy inverso, optimización de rendimiento y seguridad de aplicaciones.	Escalabilidad y rendimiento a nivel empresarial, configuración avanzada de políticas de tráfico, balanceo de carga inteligente y adaptativo, integración con sistemas de seguridad, gestión unificada de la entrega de aplicaciones, monitoreo y análisis avanzados, alta disponibilidad y seguridad.	Ambos

18.21. Tecnologías sugeridas: Escalabilidad vertical y horizontal

Herramienta/Tecnología	Descripción	Beneficios	Cloud (Cuál)/OnPremise
Apache Kafka	Plataforma de mensajería distribuida diseñada para manejar flujos de datos en tiempo real. Permite escalar tanto vertical como horizontalmente mediante la adición de nodos y particiones.	Alta escalabilidad tanto vertical como horizontal, procesamiento distribuido y paralelo, tolerancia a fallos, replicación y particionamiento de datos, soporte para flujos de datos de alto volumen y velocidad, integración con sistemas existentes, monitorización y gestión centralizadas.	Ambos

Active MQ	Sistema de mensajería de código abierto que implementa el protocolo Advanced Message Queuing Protocol (AMQP). Permite escalar verticalmente añadiendo recursos a los nodos existentes y horizontalmente mediante la implementación de clústeres.	Escalabilidad vertical mediante la adición de recursos (CPU, memoria, etc.), escalabilidad horizontal mediante la creación de clústeres, soporte para diversos protocolos de mensajería, alta disponibilidad y tolerancia a fallos, gestión avanzada de colas y mensajes, integración con sistemas existentes, monitorización y alertas.	Ambos
Apache Pulsar	Plataforma de mensajería y procesamiento de datos distribuida y tolerante a fallos que permite la escalabilidad horizontal de forma nativa mediante la adición de nodos de forma independiente en clústeres separados para procesamiento y almacenamiento.	Escalabilidad horizontal nativa con particionamiento automático y balanceo de carga, procesamiento distribuido y paralelo, tolerancia a fallos y alta disponibilidad, almacenamiento persistente y replicación de datos, integración con servicios en la nube y sistemas existentes, herramientas de monitorización y administración, compatibilidad con varios protocolos de mensajería y API de cliente.	Ambos
NATS	Sistema de mensajería y streaming de código abierto que ofrece una arquitectura ligera y de alto rendimiento. NATS se escala horizontalmente mediante la implementación de clústeres y la adición de nodos según sea necesario.	Escalabilidad horizontal mediante clústeres y distribución de carga, alta velocidad y latencia baja, uso eficiente de recursos, soporte para varios modelos de mensajería (publish/subscribe, request/reply), tolerancia a fallos, seguridad avanzada, integración con sistemas existentes, monitorización y alertas.	Ambos
Amazon Kinesis	Plataforma de streaming de datos de AWS que permite procesar y analizar datos en tiempo real. Kinesis proporciona escalabilidad horizontal automática al ajustar automáticamente la capacidad de streaming en función del volumen de datos.	Escalabilidad horizontal automática y dinámica, procesamiento de datos en tiempo real, integración con servicios de AWS, almacenamiento duradero y replicación, seguridad y cumplimiento normativo, fácil integración con herramientas de análisis y procesamiento, herramientas de monitorización y alertas.	Cloud (AWS)

Azure Event Hubs	Plataforma de streaming de eventos gestionada por Azure que puede procesar y analizar grandes volúmenes de datos de forma escalable. Azure Event Hubs ofrece particionamiento automático y balanceo de carga para escalar horizontalmente y manejar cargas de trabajo de alto rendimiento.	Escalabilidad horizontal automática mediante particionamiento y balanceo de carga, procesamiento de eventos en tiempo real, integración con servicios de Azure, almacenamiento duradero y replicación, seguridad avanzada, fácil integración con herramientas de análisis y procesamiento, herramientas de monitorización y alertas.	Cloud (Azure)
Google Cloud Pub/Sub	Servicio de mensajería y publicación/suscripción de Google Cloud que ofrece escalabilidad horizontal automática para manejar cargas de trabajo de cualquier tamaño.	Escalabilidad horizontal automática y global, procesamiento de mensajes en tiempo real, integración con servicios de Google Cloud, almacenamiento duradero y replicación, seguridad avanzada, fácil integración con herramientas de análisis y procesamiento, herramientas de monitorización y alertas.	Cloud (Google Cloud)

18.22. Tecnologías sugeridas: Protocolos de comunicación

Herramienta/Tecnología	Descripción	Beneficios	Cloud (Cuál)/OnPremise
MQTT (Message Queuing Telemetry Transport)	Protocolo ligero de mensajería para dispositivos IoT y aplicaciones con restricciones de ancho de banda y consumo de energía. MQTT es ampliamente utilizado en aplicaciones de IoT debido a su simplicidad, eficiencia y soporte para publicación/suscripción de mensajes.	Eficiente en el uso de ancho de banda y consumo de energía, simple y fácil de implementar, soporte para la publicación/suscripción de mensajes, conexiones persistentes y fiables, escalabilidad para grandes cantidades de dispositivos, seguridad y cifrado de extremo a extremo.	Ambos

AMQP (Advanced Message Queuing Protocol)	Protocolo de mensajería de nivel empresarial diseñado para aplicaciones de negocio y sistemas de mensajería distribuida. AMQP proporciona funcionalidades avanzadas como colas de mensajes, enrutamiento y aseguramiento de la entrega de mensajes.	Rico en funcionalidades para aplicaciones empresariales, soporte para colas de mensajes y enrutamiento avanzado, garantía de entrega de mensajes, interoperabilidad entre diferentes plataformas y lenguajes de programación, seguridad y autenticación integradas.	Ambos
STOMP (Simple Text Oriented Messaging Protocol)	Protocolo simple y fácil de implementar para la comunicación entre clientes y servidores de mensajería. STOMP se utiliza comúnmente en aplicaciones web y sistemas de mensajería donde la simplicidad es prioritaria.	Fácil de implementar y entender, soporte para diversas plataformas y lenguajes de programación, interoperabilidad entre diferentes sistemas de mensajería, uso eficiente de recursos, ampliamente utilizado en aplicaciones web y sistemas de mensajería basados en texto.	Ambos
CoAP (Constrained Application Protocol)	Protocolo diseñado para aplicaciones IoT y dispositivos con recursos limitados, como sensores y actuadores. CoAP ofrece una comunicación eficiente y ligera mediante la transferencia de mensajes a través de UDP y soporte para operaciones CRUD (Create, Read, Update, Delete).	Eficiente en el uso de recursos y ancho de banda, diseñado para dispositivos con recursos limitados, soporte para operaciones CRUD, interoperabilidad con HTTP, seguridad y cifrado integrados, escalabilidad para grandes redes de dispositivos IoT.	Ambos
gRPC (Google Remote Procedure Call)	Sistema de llamada a procedimiento remoto (RPC) de código abierto desarrollado por Google que utiliza HTTP/2 como protocolo de transporte y Protocol Buffers como formato de serialización de datos. gRPC es eficiente, de alto rendimiento y adecuado para aplicaciones distribuidas y servicios en la nube.	Alta eficiencia y rendimiento, soporte para varios lenguajes de programación, generación automática de código cliente y servidor, seguridad integrada con TLS/SSL, streaming bidireccional y transmisión de datos de alta velocidad, integración con balanceadores de carga y herramientas de monitorización.	Ambos

WebSockets	Protocolo de comunicación bidireccional y de bajo nivel que permite la comunicación interactiva en tiempo real entre clientes y servidores a través de una conexión TCP persistente. WebSockets se utiliza comúnmente en aplicaciones web y servicios de mensajería en tiempo real.	Comunicación bidireccional y en tiempo real, bajo consumo de recursos, establecimiento de conexiones persistentes, compatibilidad con múltiples lenguajes y plataformas, ampliamente soportado por navegadores web y servidores, adecuado para aplicaciones web interactivas y servicios de mensajería en tiempo real.	Ambos
HTTP/HTTPS	Protocolo de transferencia de hipertexto ampliamente utilizado para la comunicación entre clientes y servidores en la web. HTTP/HTTPS proporciona una comunicación basada en solicitudes y respuestas, con soporte para diversos métodos de solicitud y seguridad mediante el cifrado SSL/TLS.	Ampliamente compatible y soportado, uso sencillo a través de API RESTful, seguridad mediante cifrado SSL/TLS, interoperabilidad entre diferentes sistemas y plataformas, escalabilidad y rendimiento mejorados con HTTP/2, adecuado para aplicaciones web, APIs y servicios RESTful.	Ambos

18.23. Componente clave: Cache

Cache	Decisiones arquitectónicas Balanceo de carga Cache Patrones Shared filesystem

18.24. Tecnologías sugeridas: Cache

Herramienta/Tecnología	Descripción	Beneficios	Cloud (Cuál)/OnPremise
Redis	Redis es una base de datos en memoria de código abierto que se utiliza comúnmente como caché, almacén de datos en caché y almacén de valores clave. Ofrece una estructura de datos versátil y de alto rendimiento, como cadenas, listas, conjuntos y mapas ordenados, y proporciona capacidades avanzadas como la persistencia de datos, la replicación y la alta disponibilidad.	Alta velocidad y rendimiento, baja latencia, estructuras de datos flexibles, persistencia de datos opcional, escalabilidad horizontal mediante clústeres, soporte para caché distribuida y particionamiento, integración con varios lenguajes de programación y marcos, compatibilidad con transacciones y operaciones atómicas.	Ambos
Memcached	Memcached es un sistema de almacenamiento en caché de objetos distribuido de código abierto que almacena datos en memoria para mejorar el rendimiento de las aplicaciones. Es ampliamente utilizado para cachear resultados de bases de datos, consultas HTTP y datos generados dinámicamente.	Alta velocidad y rendimiento, baja latencia, escalabilidad horizontal mediante clústeres, distribución de carga automática, manejo eficiente de grandes volúmenes de datos, fácil de usar e implementar, integración con varios lenguajes de programación y marcos, caché distribuida y particionamiento.	Ambos
Apache Ignite	Apache Ignite es una plataforma de procesamiento de datos en memoria distribuida y en caché que proporciona almacenamiento en caché, procesamiento distribuido y análisis en tiempo real en una sola solución. Es altamente escalable y se puede integrar fácilmente con aplicaciones existentes a través de APIs y conectores.	Almacenamiento y procesamiento en memoria distribuida, alta velocidad y rendimiento, baja latencia, escalabilidad horizontal y vertical, integración con varios lenguajes y marcos, soporte para procesamiento de transacciones ACID, consultas SQL y análisis en tiempo real, persistencia opcional de datos, herramientas de monitorización y gestión.	Ambos

Hazelcast	Hazelcast es una plataforma de computación en memoria de código abierto que ofrece almacenamiento en caché distribuido, procesamiento distribuido y computación en memoria distribuida. Se utiliza para mejorar el rendimiento de aplicaciones distribuidas y proporcionar acceso a datos en memoria de alta velocidad.	Almacenamiento en caché distribuido y en memoria, procesamiento y computación distribuida, alta velocidad y rendimiento, baja latencia, escalabilidad horizontal y vertical, integración con varios lenguajes de programación y marcos, persistencia de datos opcional, tolerancia a fallos y alta disponibilidad, herramientas de monitorización y gestión.	Ambos
Couchbase	Couchbase es una base de datos en memoria y en caché de alto rendimiento y escalable que proporciona almacenamiento, indexación y consulta de datos JSON. Se utiliza para aplicaciones de misión crítica que requieren acceso rápido a datos en memoria y alta disponibilidad.	Almacenamiento y recuperación de datos en memoria y en caché, alta velocidad y rendimiento, baja latencia, escalabilidad horizontal y vertical, integración con varios lenguajes y marcos, soporte para consultas SQL y N1QL, persistencia de datos opcional, tolerancia a fallos y alta disponibilidad, herramientas de monitorización y gestión.	Ambos
AWS ElastiCache	AWS ElastiCache es un servicio de almacenamiento en caché completamente administrado que facilita el despliegue y la administración de cachés en la nube. Soporta tanto Redis como Memcached como motores de caché y ofrece escalabilidad automática, alta disponibilidad y seguridad integrada para mejorar el rendimiento de las aplicaciones.	Servicio completamente administrado, escalabilidad automática, alta disponibilidad y durabilidad, seguridad y cifrado de datos, compatibilidad con Redis y Memcached, integración con otros servicios de AWS, monitorización y alertas, fácil escalado y administración a través de la consola de AWS.	Cloud (AWS)
Azure Cache for Redis	Azure Cache for Redis es un servicio de caché totalmente administrado y compatible con Redis que proporciona almacenamiento en caché en la nube de alto rendimiento y baja latencia para mejorar el	Servicio completamente administrado, escalabilidad automática, alta disponibilidad y durabilidad, seguridad y cifrado de datos, compatibilidad total con Redis, integración con otros servicios de Azure,	Cloud (Azure)

	rendimiento de las aplicaciones. Ofrece escalabilidad automática, alta disponibilidad y seguridad integrada para simplificar la implementación y gestión de cachés en Azure.	monitorización y alertas, fácil escalado y administración a través del portal de Azure.	
Google Cloud Memorystore	Google Cloud Memorystore es un servicio de almacenamiento en caché completamente administrado que ofrece compatibilidad con Redis para almacenar y gestionar datos en memoria en la nube. Proporciona escalabilidad automática, alta disponibilidad y seguridad integrada para mejorar el rendimiento de las aplicaciones en Google Cloud Platform.	Servicio completamente administrado, escalabilidad automática, alta disponibilidad y durabilidad, seguridad y cifrado de datos, compatibilidad total con Redis, integración con otros servicios de Google Cloud, monitorización y alertas, fácil escalado y administración a través de la consola de Google Cloud.	Cloud (Google Cloud)

18.25. Componente clave: Persistencia

Persistencia	Decisiones arquitectónicas Base de datos relacional Base de datos No-SQL

18.26. Tecnologías sugeridas: Bases de datos relacionales

Herramienta/Tecnología	Descripción	Beneficios	Cloud (Cuál)/OnPremise
PostgreSQL	PostgreSQL es un sistema de gestión de bases de datos relacional de código abierto y altamente extensible. Ofrece soporte para SQL completo, transacciones ACID, integridad referencial, replicación, particionamiento de tablas y una amplia gama de extensiones y características avanzadas.	Confiabilidad y estabilidad, cumplimiento de estándares SQL, soporte para transacciones ACID, escalabilidad y rendimiento, amplia gama de extensiones y características, alta flexibilidad y personalización, comunidad activa de usuarios y desarrolladores.	Ambos

MySQL	MySQL es una base de datos relacional de código abierto ampliamente utilizada en aplicaciones web y empresariales. Ofrece soporte para SQL completo, transacciones ACID, replicación, particionamiento de tablas, funciones avanzadas de seguridad y una comunidad activa de usuarios y desarrolladores.	Ampliamente utilizado y probado en aplicaciones web y empresariales, escalabilidad y rendimiento, soporte para transacciones ACID, replicación y particionamiento, funciones avanzadas de seguridad, flexibilidad y extensibilidad, comunidad activa de usuarios y desarrolladores.	Ambos
Oracle Database	Oracle Database es un sistema de gestión de bases de datos relacional líder en la industria, conocido por su escalabilidad, rendimiento y seguridad. Ofrece una amplia gama de características empresariales, incluyendo soporte para SQL completo, transacciones ACID, clustering, particionamiento y replicación.	Escalabilidad, rendimiento y fiabilidad empresarial, seguridad avanzada y cumplimiento, amplia gama de características y opciones, soporte para SQL completo y transacciones ACID, herramientas de gestión y monitorización integradas, servicio de soporte técnico de nivel empresarial.	Ambos
Microsoft SQL Server	Microsoft SQL Server es un sistema de gestión de bases de datos relacional desarrollado por Microsoft. Ofrece soporte para SQL completo, transacciones ACID, replicación, particionamiento de tablas, procedimientos almacenados y funciones avanzadas de seguridad. Es especialmente popular en entornos Windows y con integración estrecha con otras herramientas de Microsoft.	Integración estrecha con el ecosistema de Microsoft, escalabilidad y rendimiento, seguridad avanzada y cumplimiento, soporte para transacciones ACID, procedimientos almacenados y funciones de programación, herramientas de gestión y monitorización integradas, opciones de implementación flexibles.	Ambos
Amazon RDS for PostgreSQL	Amazon RDS for PostgreSQL es un servicio de bases de datos relacionales administrado y escalable que ejecuta instancias de PostgreSQL en la nube de AWS. Ofrece escalabilidad automática, respaldos automáticos, alta disponibilidad y seguridad integrada para simplificar la implementación y gestión de bases de datos PostgreSQL en la nube.	Servicio administrado y escalable, escalabilidad automática y alta disponibilidad, seguridad y cifrado de datos, respaldos automáticos y recuperación ante desastres, integración con otros servicios de AWS, opciones flexibles de implementación y facturación basada en el uso.	Cloud (AWS)

Google Cloud SQL for MySQL	Google Cloud SQL for MySQL es un servicio de bases de datos relacionales completamente administrado que ejecuta instancias de MySQL en la nube de Google Cloud. Ofrece escalabilidad automática, alta disponibilidad, seguridad integrada y opciones flexibles de implementación para aplicaciones en la nube.	Servicio completamente administrado, escalabilidad automática y alta disponibilidad, seguridad y cifrado de datos, integración con otros servicios de Google Cloud, opciones flexibles de implementación, monitoreo y alertas integradas.	Cloud (Google Cloud)
Azure SQL Database	Azure SQL Database es un servicio de bases de datos relacionales administrado y escalable que se ejecuta en la nube de Microsoft Azure. Ofrece escalabilidad automática, alta disponibilidad, seguridad avanzada y compatibilidad con SQL Server para simplificar la implementación y gestión de bases de datos en la nube.	Servicio administrado y escalable, escalabilidad automática y alta disponibilidad, seguridad avanzada y cumplimiento, compatibilidad con SQL Server, integración con otros servicios de Azure, opciones flexibles de implementación y facturación basada en el uso.	Cloud (Azure)

18.27. Tecnologías sugeridas: Bases de datos No-SQL

Herramienta/Tecnología	Descripción	Beneficios	Cloud (Cuál)/OnPremise
MongoDB	MongoDB es una base de datos NoSQL de documentos de código abierto que utiliza un modelo de datos flexible basado en documentos JSON. Ofrece escalabilidad horizontal, alta disponibilidad, indexación rica y consultas poderosas para manejar una variedad de casos de uso, desde aplicaciones web hasta analítica en tiempo real.	Modelo de datos flexible y semiestructurado, escalabilidad horizontal y alta disponibilidad, consultas potentes y ricas, indexación eficiente, soporte para replicación y sharding, integración con lenguajes de programación y marcos populares, comunidad activa de usuarios y desarrolladores.	Ambos
Apache Cassandra	Apache Cassandra es una base de datos NoSQL distribuida y altamente escalable diseñada para manejar grandes volúmenes de datos y cargas de trabajo distribuidas. Utiliza un modelo de datos basado	Escalabilidad lineal y alta disponibilidad, tolerancia a fallos y resistencia, modelo de datos basado en columnas, rendimiento de escritura y lectura rápido, soporte para replicación y particionamiento	Ambos

	en columnas y ofrece alta disponibilidad, tolerancia a fallos y rendimiento linealmente escalable para aplicaciones que requieren escalabilidad global y disponibilidad continua.	automático, integración con lenguajes de programación y marcos, flexibilidad en el esquema de datos.	
Couchbase	Couchbase es una base de datos NoSQL distribuida y en memoria que combina almacenamiento en caché, indexación y consulta de datos JSON. Ofrece escalabilidad horizontal, alta disponibilidad, rendimiento de submilisegundo y soporte para transacciones ACID, lo que la hace adecuada para aplicaciones web, móviles y de Internet de las cosas (IoT).	Almacenamiento y consulta de datos JSON en memoria, escalabilidad horizontal y alta disponibilidad, rendimiento de submilisegundo, soporte para transacciones ACID, integración con lenguajes de programación y marcos populares, herramientas de monitorización y gestión integradas.	Ambos
Amazon DynamoDB	Amazon DynamoDB es un servicio de base de datos NoSQL completamente administrado que ofrece almacenamiento en clave-valor y documentos para aplicaciones web y móviles a cualquier escala. Proporciona escalabilidad automática, rendimiento de milisegundos y seguridad integrada para simplificar la implementación y gestión de bases de datos en la nube.	Servicio completamente administrado, escalabilidad automática y rendimiento rápido, seguridad y cifrado de datos, integración con otros servicios de AWS, opciones flexibles de implementación, monitorización y alertas integradas.	Cloud (AWS)
Google Cloud Firestore	Google Cloud Firestore es un servicio de base de datos NoSQL completamente administrado que ofrece almacenamiento en tiempo real y sincronización de datos para aplicaciones web y móviles. Proporciona escalabilidad automática, sincronización en tiempo real, consultas flexibles y seguridad integrada para simplificar el desarrollo de aplicaciones.	Servicio completamente administrado, escalabilidad automática y sincronización en tiempo real, consultas flexibles y rápidas, seguridad y cifrado de datos, integración con otros servicios de Google Cloud, monitorización y alertas integradas.	Cloud (Google Cloud)
Azure Cosmos DB	Azure Cosmos DB es un servicio de base de datos multimodal y distribuido	Escalabilidad global y alta disponibilidad, soporte para varios modelos de	Cloud (Azure)

	globalmente que ofrece soporte para varios modelos de datos, incluidos documentos, gráficos, clave-valor y columnares. Proporciona escalabilidad global, baja latencia, alta disponibilidad y coherencia garantizada para aplicaciones distribuidas a gran escala.	datos, baja latencia y coherencia garantizada, integración con otros servicios de Azure, seguridad y cifrado de datos, opciones flexibles de implementación, monitorización y alertas integradas.	
Redis	Redis es una base de datos en memoria de código abierto que se utiliza comúnmente como base de datos NoSQL, almacén de caché y cola de mensajes. Ofrece estructuras de datos flexibles, alto rendimiento y durabilidad opcional para casos de uso que requieren acceso rápido a datos en memoria.	Almacenamiento y procesamiento de datos en memoria, estructuras de datos flexibles, alto rendimiento y baja latencia, escalabilidad horizontal, durabilidad opcional, integración con lenguajes de programación y marcos, herramientas de monitorización y gestión.	Ambos

18.28. Componente Clave: Contenerización, Orquestación y Registry

Herramienta/Tecnología	Descripción	Beneficios	Cloud (Cuál)/OnPremise
Docker	Docker es una plataforma de contenerización que permite empaquetar, distribuir y ejecutar aplicaciones en contenedores ligeros y portátiles. Los contenedores proporcionan una forma consistente de desarrollar, implementar y administrar aplicaciones, asegurando que se ejecuten de manera idéntica en cualquier entorno, ya sea local, en la nube o en un centro de datos.	Aislamiento de aplicaciones, portabilidad entre entornos, eficiencia de recursos, velocidad de implementación y escalabilidad, gestión simplificada de infraestructura, fácil integración con otras herramientas y tecnologías.	Ambos

Kubernetes	Kubernetes es una plataforma de orquestación de contenedores de código abierto que automatiza la implementación, el escalado y la gestión de aplicaciones en contenedores. Proporciona una infraestructura escalable y de alta disponibilidad para ejecutar cargas de trabajo en contenedores, simplificando las operaciones de la aplicación y permitiendo una implementación rápida y flexible en cualquier entorno, desde máquinas virtuales hasta entornos de nube híbrida.	Automatización de implementaciones y escalado, gestión centralizada de clústeres, auto recuperación de fallos, actualizaciones sin tiempo de inactividad, monitoreo y registro integrados, soporte para diferentes tipos de cargas de trabajo, ecosistema de herramientas y extensiones amplio y activo.	Ambos
Docker Registry (Docker Hub)	Docker Registry es un servicio de almacenamiento de imágenes de contenedores que permite a los usuarios almacenar y distribuir imágenes de contenedores de Docker. Docker Hub es un registro público gratuito que ofrece una amplia variedad de imágenes de contenedores listas para usar, así como herramientas para construir, probar y desplegar aplicaciones en contenedores. También permite a los usuarios crear y compartir imágenes personalizadas de contenedores para sus propias aplicaciones.	Acceso a una amplia biblioteca de imágenes de contenedores, facilidad para compartir imágenes personalizadas, integración con Docker Engine y otras herramientas de desarrollo, características de seguridad y control de acceso, capacidades de almacenamiento y gestión de versiones.	Ambos
Rancher	Rancher es una plataforma de gestión de contenedores de código abierto que facilita la implementación, la administración y la operación de clústeres de Kubernetes. Proporciona una interfaz gráfica de usuario intuitiva y herramientas de automatización para simplificar tareas como el aprovisionamiento de clústeres, la implementación de	Interfaz de usuario intuitiva, automatización de tareas complejas, gestión centralizada de múltiples clústeres, herramientas de monitoreo y alerta, integración con plataformas de nube y herramientas de CI/CD, soporte para implementaciones locales y en la nube.	Ambos

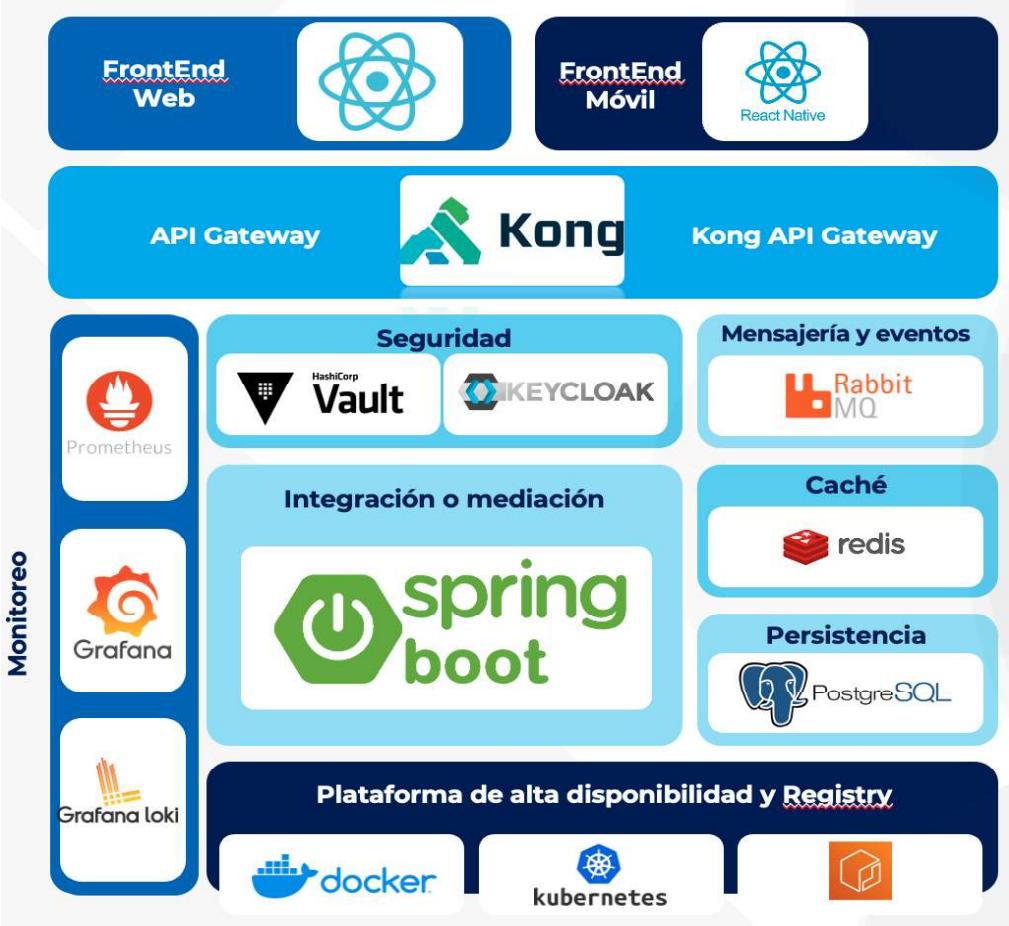
	aplicaciones y la administración del ciclo de vida de los contenedores.		
OpenShift	OpenShift es una plataforma de aplicaciones de código abierto basada en Kubernetes que ofrece una experiencia de desarrollo, implementación y administración de aplicaciones integrada. Proporciona herramientas de desarrollo, automatización de implementaciones, gestión de clústeres y servicios de plataforma como servicio (PaaS) para acelerar el ciclo de vida de las aplicaciones y facilitar la entrega continua.	Experiencia integrada de desarrollo y operaciones, automatización de implementaciones y gestión de clústeres, integración con herramientas de CI/CD, soporte para contenedores y aplicaciones tradicionales, seguridad integrada y control de acceso, escalabilidad y alta disponibilidad, flexibilidad para implementaciones locales y en la nube.	Ambos

Se puede llegar a la conclusión que existen muchas tecnologías y herramientas para resolver los atributos de calidad y requerimientos no funcionales, la elección de las tecnologías y herramientas se llevó a cabo con el grupo de arquitectos de Catastro teniendo en cuenta las herramientas que ya se tenían implementadas, en las cuales se tenía una alta experiencia, las que se encontraban habilitadas en los diferentes contratos de TI y las que aunque no estaban en ninguno de los grupos anteriormente mencionados aportan valor a la arquitectura.

19. Arquitectura de referencia y sus tecnologías

Después de haber estructurado la arquitectura de referencia propuesta la cual fue diseñada para optimizar la escalabilidad, la seguridad y la eficiencia operativa en un entorno distribuido tanto en Nube como en OnPremise.

Este capítulo detalla cada una de las tecnologías seleccionadas y los beneficios de las mismas para cumplir con los requerimientos no funcionales de la plataforma de catastro virtual.



19.1. React.js (Frontend Web)

- Propósito:
  - Desarrollar una interfaz de usuario interactiva y dinámica para el portal web de la plataforma.
- Beneficios:
  - Facilita el desarrollo de aplicaciones de una sola página (SPA) que mejoran la experiencia del usuario.
  - Promueve la reutilización de componentes, lo que acelera el desarrollo y mantenimiento.
  - Tiene un ecosistema robusto y una gran comunidad de soporte.

19.2. React Native (Frontend Móvil)

Propósito:

- Construir aplicaciones móviles nativas para iOS y Android desde un único código base.

Beneficios:

- Reduce los costos y tiempos de desarrollo al reutilizar código para múltiples plataformas.
- Proporciona una experiencia nativa optimizada para los usuarios finales.
- Permite iteraciones rápidas gracias a herramientas como Hot Reloading.

19.3. Kong Gateway (API Gateway)

Propósito:

- Gestionar y enrutar las solicitudes entre los clientes (web/móvil) y los microservicios backend.

Beneficios:

- Centraliza la seguridad, autenticación y autorización de las APIs.
- Ofrece monitoreo, limitación de tasas (rate limiting) y transformaciones de datos.
- Simplifica la integración con sistemas externos y el manejo de servicios.

19.4. Spring Boot (Backend y Microservicios)

Propósito:

- Proveer una plataforma para desarrollar microservicios robustos y escalables en Java.

Beneficios:

- Simplifica el desarrollo con configuraciones predeterminadas inteligentes.
- Integra herramientas de Spring Cloud para orquestación, configuración y resiliencia.
- Ofrece soporte para CI/CD y despliegues rápidos.

19.5. Keycloak (Gestión de Autenticación y Autorización)

Propósito:

- Administrar la autenticación de usuarios, el inicio de sesión único (SSO) y la gestión de roles y permisos.

Beneficios:

- Soporta protocolos estándar como OAuth2, OpenID Connect y SAML.
- Simplifica la gestión de usuarios con características como federación de identidad.
- Mejora la seguridad y facilita la integración con otras aplicaciones.

19.6. HashiCorp Vault (Gestión de Secretos)

Propósito:

- Proteger y gestionar credenciales, claves de API y otros secretos necesarios para el sistema.

Beneficios:

- Centraliza el almacenamiento seguro de secretos, reduciendo riesgos de exposición.
- Ofrece control granular de acceso a secretos según roles y permisos.
- Facilita la rotación de claves y contraseñas para mitigar riesgos.

19.7. RabbitMQ (Mensajería y Gestión de Eventos)

Propósito:

- Gestionar la comunicación asincrónica entre los microservicios mediante colas de mensajes.

Beneficios:

- Mejora la resiliencia al desacoplar los servicios.
- Optimiza el rendimiento en operaciones que no requieren respuestas inmediatas.
- Permite la implementación de patrones como "publicar y suscribir" para notificaciones.

19.8. Redis (Caché)

Propósito:

- Almacenar en memoria datos de uso frecuente para reducir la latencia y mejorar el rendimiento.

Beneficios:

- Aumenta la velocidad de acceso a datos críticos como configuraciones y resultados de consultas.
- Permite el manejo de sesiones distribuidas y conteo en tiempo real.
- Soporta estructuras avanzadas como listas, conjuntos y mapas.

19.9. PostgreSQL (Base de Datos Relacional)

Propósito:

- Almacenar y gestionar datos estructurados del sistema, incluyendo información catastral y transaccional.

Beneficios:

- Soporte nativo para datos espaciales a través de PostGIS, ideal para mapas interactivos.
- Alta confiabilidad y consistencia en transacciones.
- Compatible con herramientas modernas de análisis y reporting.

19.10. Prometheus (Monitoreo)

Propósito:

- Recolectar métricas del sistema en tiempo real para supervisar el rendimiento.

Beneficios:

- Ofrece una base de datos de series temporales optimizada para métricas.
- Permite configurar alertas basadas en reglas específicas.
- Se integra fácilmente con herramientas como Grafana para visualización.

19.11. Grafana (Visualización)

Propósito:

- Crear paneles interactivos para visualizar métricas y logs del sistema.

Beneficios:

- Mejora la observabilidad con paneles personalizados y adaptables.
- Facilita la detección de problemas mediante análisis visual en tiempo real.
- Admite múltiples fuentes de datos.

19.12. Grafana Loki (Gestión de Logs)

Propósito:

- Centralizar y gestionar los registros generados por los microservicios.

Beneficios:

- Simplifica la búsqueda de logs para depuración y auditoría.
- Integra de forma nativa con Grafana para correlacionar métricas y registros.
- Optimizado para entornos distribuidos.

19.13. Kubernetes (Orquestación de Contenedores)

Propósito:

- Administrar el despliegue, escalado y operación de los contenedores de la plataforma.

Beneficios:

- Garantiza alta disponibilidad mediante balanceo de carga y recuperación automática.
- Simplifica el escalado horizontal según la demanda.
- Ofrece soporte para configuraciones declarativas mediante archivos YAML.

19.14. Docker (Contenedores)

Propósito:

- Proveer entornos consistentes y portables para los microservicios.

Beneficios:

- Facilita el empaquetado de aplicaciones junto con sus dependencias.
- Acelera el desarrollo y despliegue en entornos locales y en producción.
- Mejora la consistencia en los entornos de desarrollo y operación.

19.15. Amazon Elastic Container Registry (ECR)

Propósito:

- Proveer un repositorio seguro y altamente disponible para almacenar, gestionar y desplegar imágenes de contenedores Docker, facilitando su integración con los flujos de trabajo de desarrollo y despliegue de la plataforma Catastro Virtual.

Beneficios:

- Gestión Centralizada de Imágenes: Permite almacenar y versionar imágenes de contenedores en un repositorio privado, garantizando un control preciso sobre las versiones de las aplicaciones.
- Seguridad Integrada: Incluye cifrado de datos en reposo mediante AWS Key Management Service (KMS) y control de acceso granular basado en IAM.
- Alta Disponibilidad: Diseñado para garantizar que las imágenes estén siempre accesibles en entornos de producción y desarrollo.
- Integración con AWS: Se integra perfectamente con otros servicios de AWS como ECS, EKS y CodePipeline, optimizando el flujo de CI/CD.
- Escalabilidad Automática: Soporta la escalabilidad en la demanda de acceso a las imágenes sin interrupciones.
- Reducción de Costos: Elimina la necesidad de gestionar infraestructura adicional para almacenar imágenes de contenedores, utilizando un modelo de pago por uso