

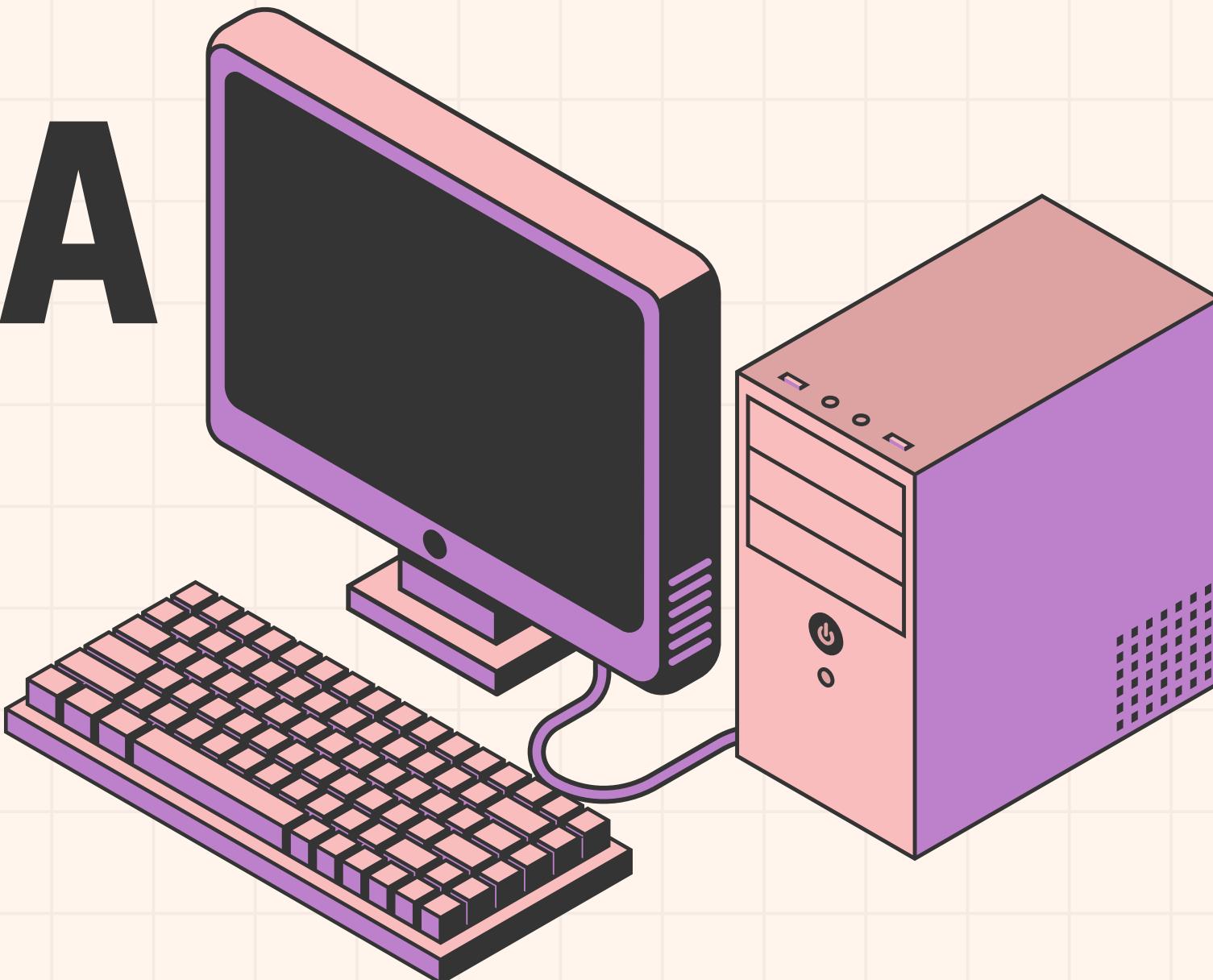
# ARQUITECTURA DE SOFTWARE

Taller Presentación 1 – Stack de Colas y Eventos

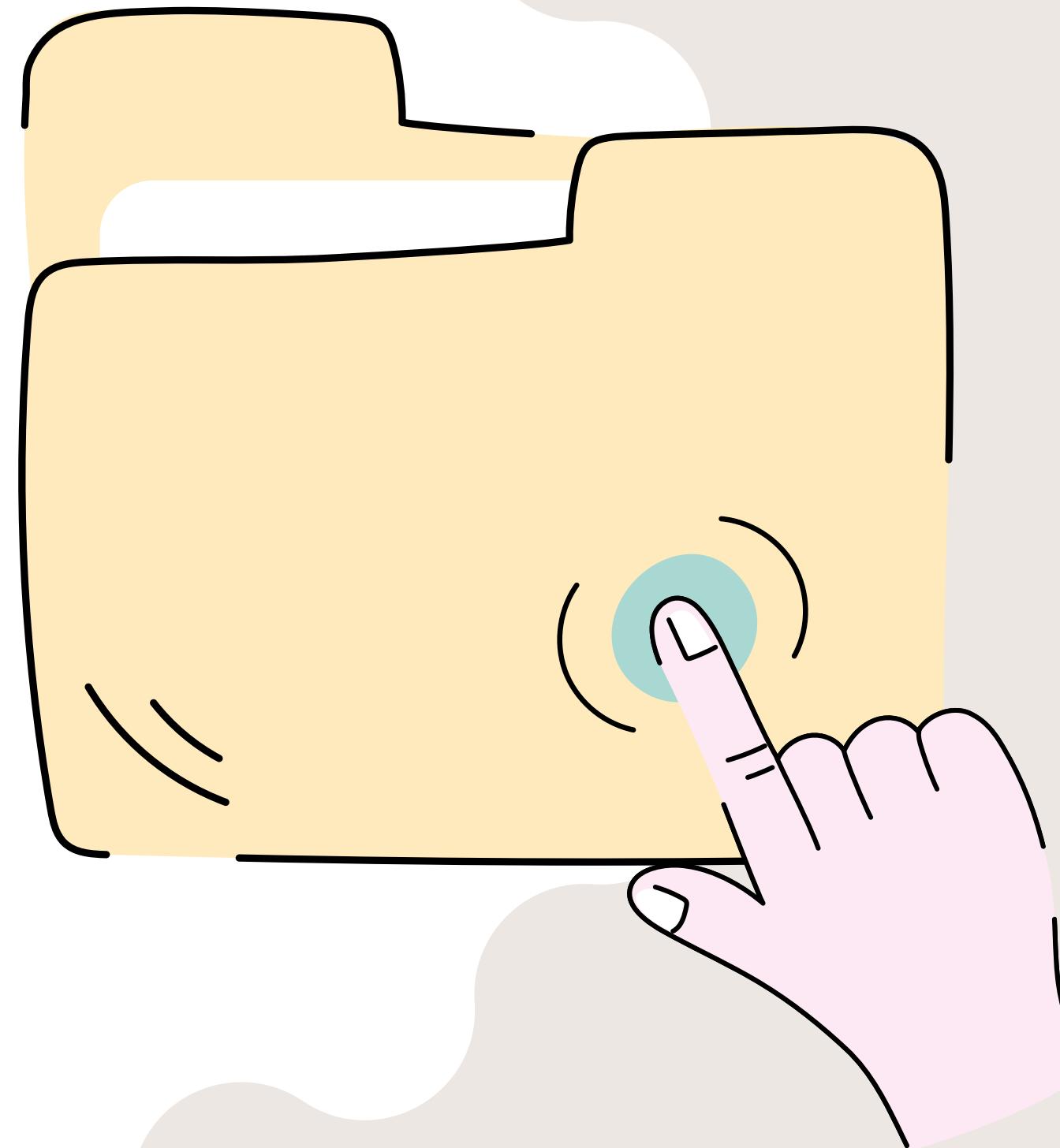
Arquitectura de Software – Grupo 7

Integrantes: Juan Diego Romero, Pablo Quintero, Kamilt Bejarano

Fecha: 6 de octubre de 2025



# Contenido



Esta presentación se enfoca en explicar un Stack para el desarrollo de aplicaciones web orientadas a eventos, compuesto de las siguientes tecnologías:

- Arquitectura Dirigida por Eventos (Event-Driven Architecture, EDA) y colas de mensajes (Message Queue Architecture)
- El lenguaje Ruby
- Los Framework Ruby on Rails y Astro.js
- El protocolo de mensajes y eventos Apache Kafka
- La base de datos Cassandra

# **Arquitecturas por Colas de mensajes**

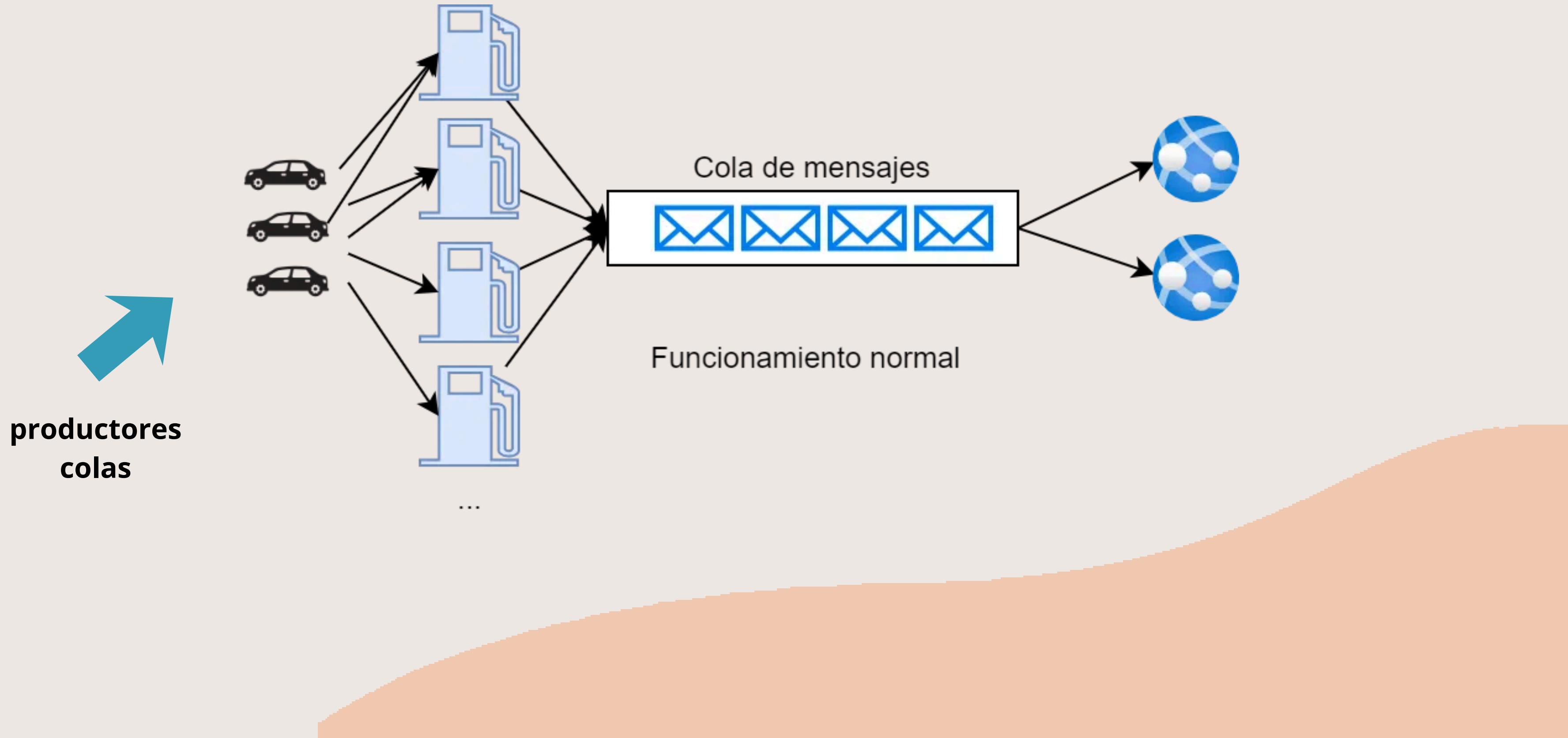
# Arquitectura de Colas de Mensajes (MQA).

***¿Qué es?***

Es la arquitectura que usa colas como buffer entre productores y consumidores, permitiendo comunicación asíncrona, confiable y escalable



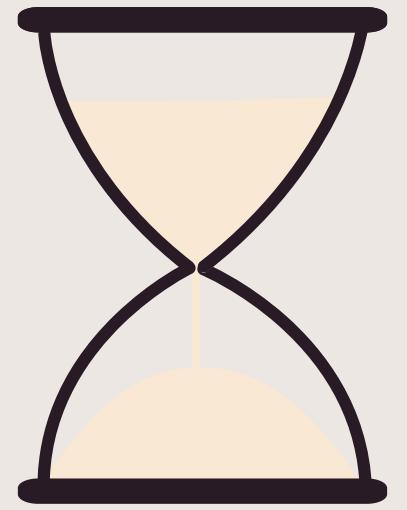
# Cómo funciona MQA





# Características de MQA

- **Comunicación** asíncrona y desacoplada
- **Persistencia** y orden de mensajes (*FIFO, Exactly-once*)
- **Tolerancia** a fallos con reintentos automáticos
- **Escalabilidad** y **balanceo** de carga con múltiples consumidores
- **Integración** con microservicios y EDA



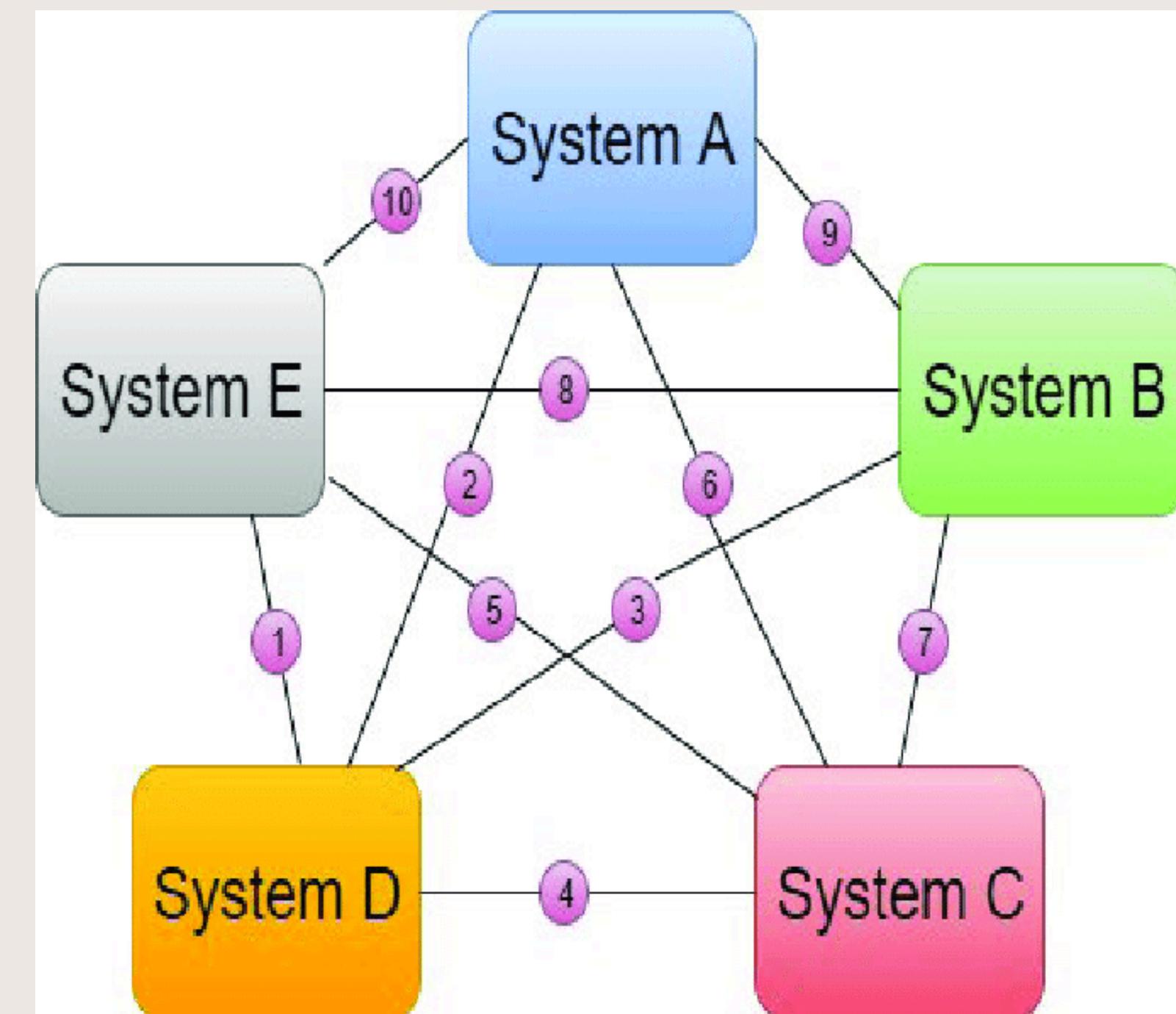
# Historia y evolución de MQA

- 📌 **1980s-1990s** → Nacen los Message-Oriented Middleware (IBM MQSeries)
- 📌 **2000s** → SOA integra colas en buses empresariales (ActiveMQ, RabbitMQ)
- 📌 **2010s** → Surgen brokers distribuidos (Kafka, Pulsar) con streaming
- 📌 **2020s** → Mensajería cloud (AWS SQS, Azure Service Bus, Google Pub/Sub)

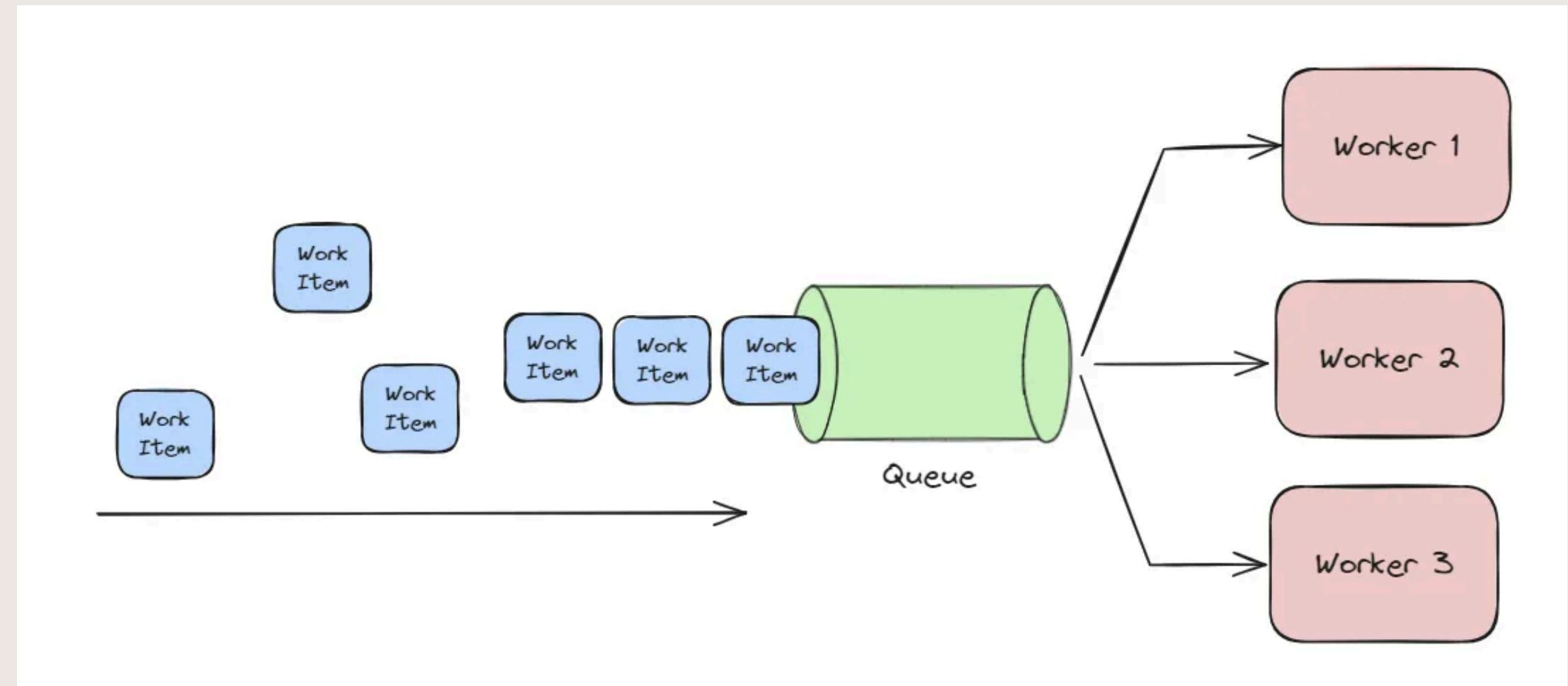
# Tipos de Arquitecturas de Colas

- **Point-to-Point (P2P)**: un receptor por mensaje (Amazon SQS, RabbitMQ)
- **Publish/Subscribe**: un mensaje llega a múltiples consumidores (Kafka, Pulsar)
- **Work Queue**: varios consumidores compiten por mensajes (Celery, Sidekiq)
- **Request-Reply**: el emisor espera una respuesta en otra cola (ActiveMQ, ZeroMQ)
- **Event Streaming**: los mensajes se almacenan en un log inmutable (Kafka, Redpanda)
- **Dead Letter Queue (DLQ)**: mensajes fallidos para auditoría o recuperación (RabbitMQ DLQ, AWS SQS DLQ)

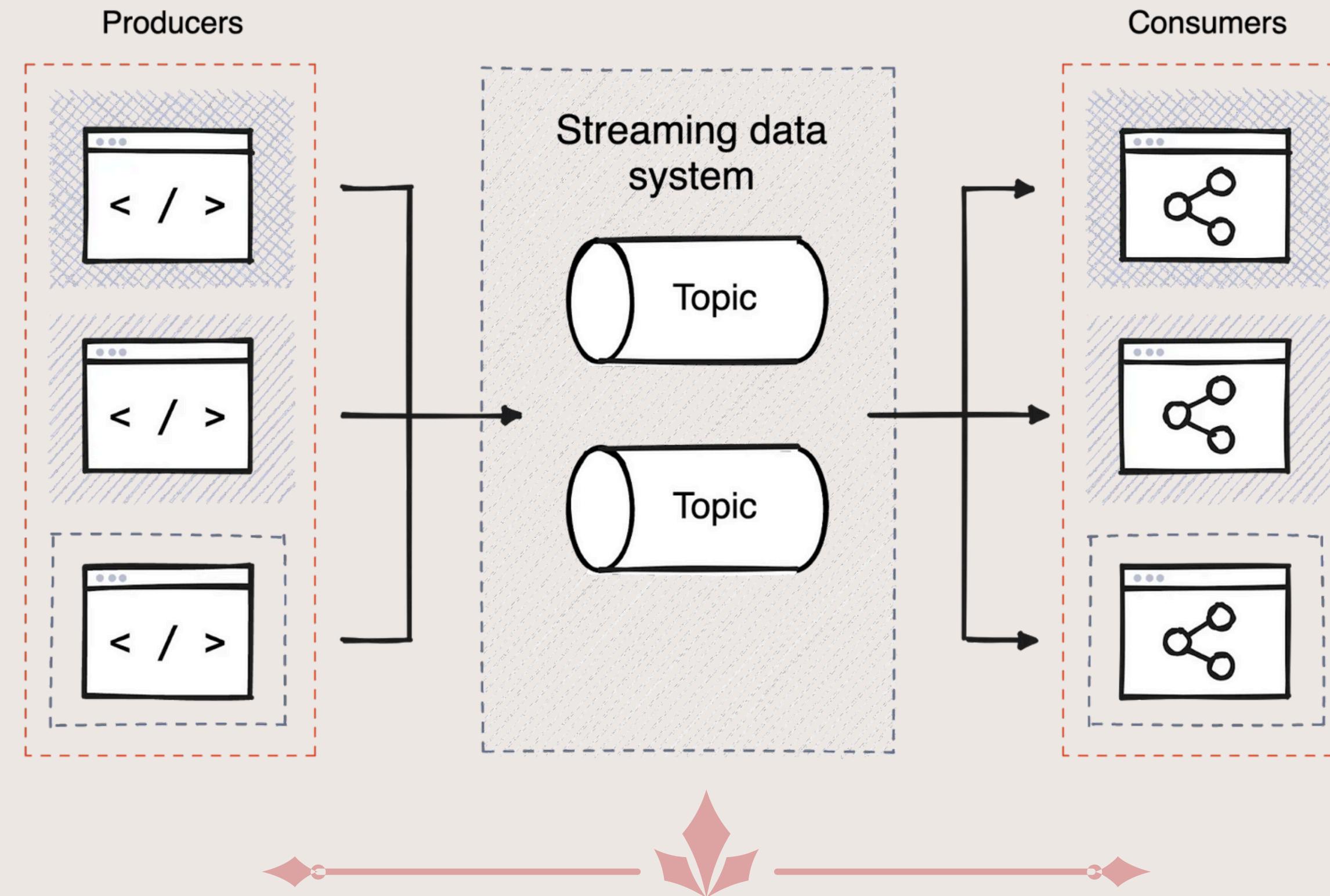
# Point-to-Point



# Work Queue



# Event Streaming

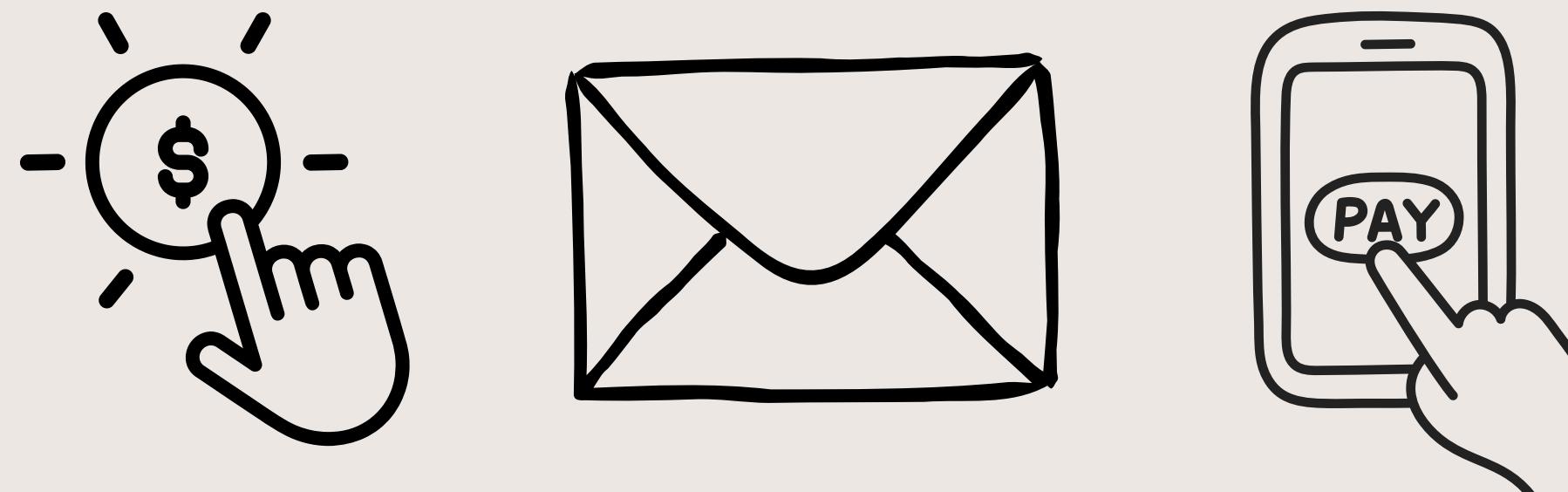


# Ventajas y Desventajas de MQA

<b>VENTAJAS</b>	<b>DESVENTAJAS</b>
Desacopla productores y consumidores	Complejidad operativa y monitoreo constante
Alta tolerancia a fallos	Latencia en el procesamiento
Escalabilidad horizontal	Possible duplicidad de mensajes
Entrega confiable incluso ante fallos	Depuración más compleja
Control de flujo y reintentos automáticos	Requiere políticas de limpieza de colas

# Casos de uso de MQA

- **Tareas en segundo plano** → emails, reportes, validaciones
- **Microservicios** → comunicación asíncrona y desacoplada
- **Sistemas de negocio** → pedidos, pagos, inventarios
- **IoT y telemetría** → ingestión de datos de sensores
- **Serverless** → ejecución de funciones disparadas por mensajes



# Casos de aplicación en la industria

	<b>Empresa</b>	<b>Implementación</b>	<b>Propósito</b>
	<b>Netflix</b>	RabbitMQ y Kafka	Orquestar operaciones y notificaciones
	<b>Uber</b>	Kafka y Redis Streams	Coordinar viajes, pagos y ubicación
	<b>Spotify</b>	Google Pub/Sub	Sincronizar listas y usuarios
	<b>Airbnb</b>	RabbitMQ	Procesamiento de reservas y alertas
	<b>Amazon</b>	SQS/SNS	Mensajería entre servicios serverless

# Principios SOLID vs Colas

Principio	Nivel de Aplicación	Justificación
<b>S – Single Responsibility</b>	Alto	Cada cola gestiona un tipo específico de mensaje con una única responsabilidad.
<b>O – Open/Closed</b>	Alto	Se pueden agregar nuevos consumidores sin modificar los productores existentes.
<b>L – Liskov Substitution</b>	Bajo	No aplica directamente a nivel arquitectónico, ya que no hay herencia entre componentes.
<b>I – Interface Segregation</b>	Alto	Cada consumidor se suscribe solo a la cola o mensajes que necesita.
<b>D – Dependency Inversion</b>	Alto	Productores y consumidores dependen de una abstracción (la cola), no entre sí.

# Mercado laboral vs Colas

Aspecto	Descripción
Demanda	Alta en empresas que implementan microservicios, cloud y sistemas de mensajería.
Roles	Backend Developer, Integration Engineer, Cloud Architect, Data Engineer.
Nivel de especialización	Medio-Alto; requiere conocimiento de concurrencia y sistemas distribuidos.
Tecnologías destacadas	RabbitMQ, Kafka, ActiveMQ, AWS SQS, Azure Service Bus, Google Pub/Sub.
Tendencia	Creciente, impulsada por el auge de arquitecturas event-driven y serverless.
Salario promedio LATAM	Entre 4 y 8 millones de pesos mensuales, dependiendo del rol y experiencia.

# Arquitectura Dirigida por Eventos

# Arquitectura Dirigida por Eventos (Event-Driven Architecture, EDA).

## **¿Qué es?**

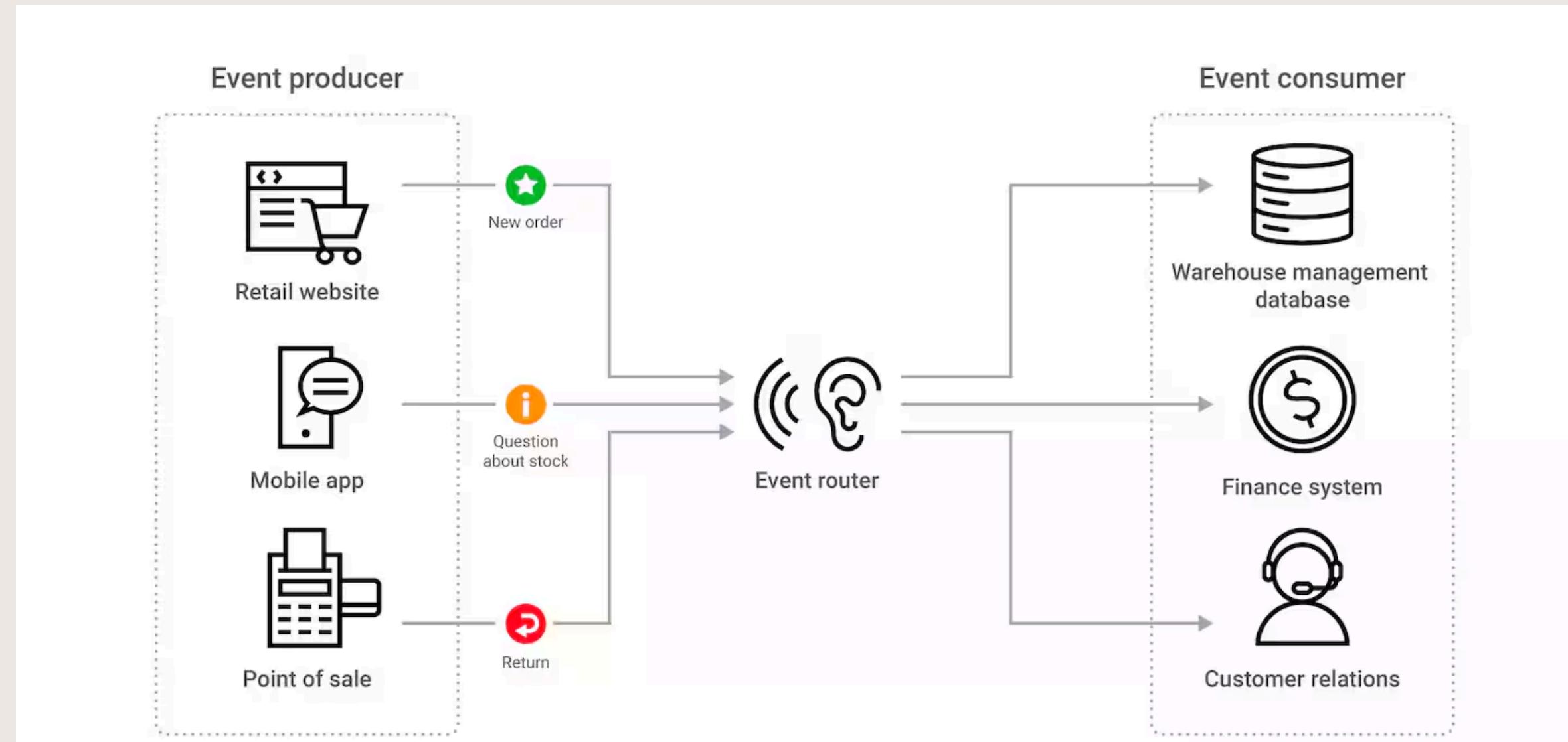
La Arquitectura Dirigida por Eventos (Event-Driven Architecture, EDA) es un modelo de diseño donde los componentes del sistema se comunican mediante eventos.

Un evento es un suceso significativo del sistema, como la creación de un pedido o la actualización de un usuario.

En EDA, los servicios no dependen directamente entre sí, sino que reaccionan a los eventos emitidos, lo que fomenta la asincronía, el desacoplamiento y la escalabilidad.



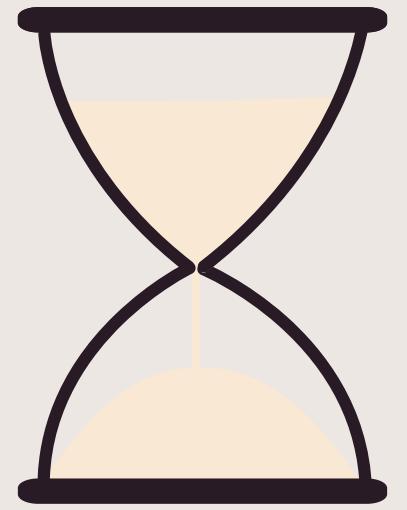
# Cómo funciona EDA





# Características de EDA

- Comunicación asíncrona y basada en eventos
- Desacoplamiento total entre productores y consumidores
- Procesamiento en tiempo real
- Escalabilidad horizontal
- Tolerancia a fallos y resiliencia
- Soporte de patrones como Publish/Subscribe, CQRS, Event Sourcing
- Consistencia eventual en sistemas distribuidos



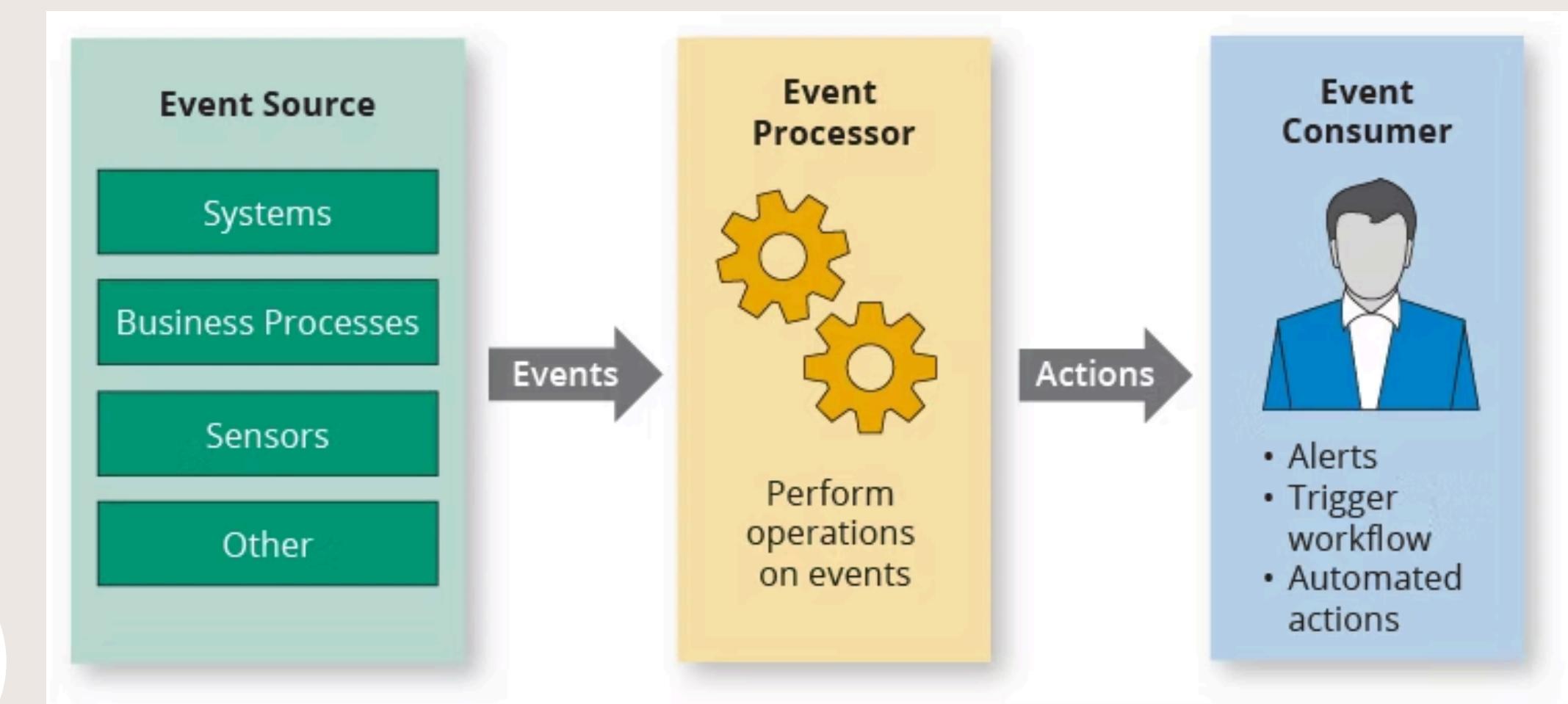
## Historia y evolución de EDA

- 📌 **Década de 2000** → Surge como evolución de la Arquitectura Orientada a Servicios (SOA).
- 📌 LinkedIn y Amazon comienzan a usar EDA para procesar datos en tiempo real.
- 📌 **2010s** → Nacen brokers como Apache Kafka, AWS SNS, Google Pub/Sub.
- 📌 **2020s** → Se consolida como base de microservicios, IoT y analítica en tiempo real.

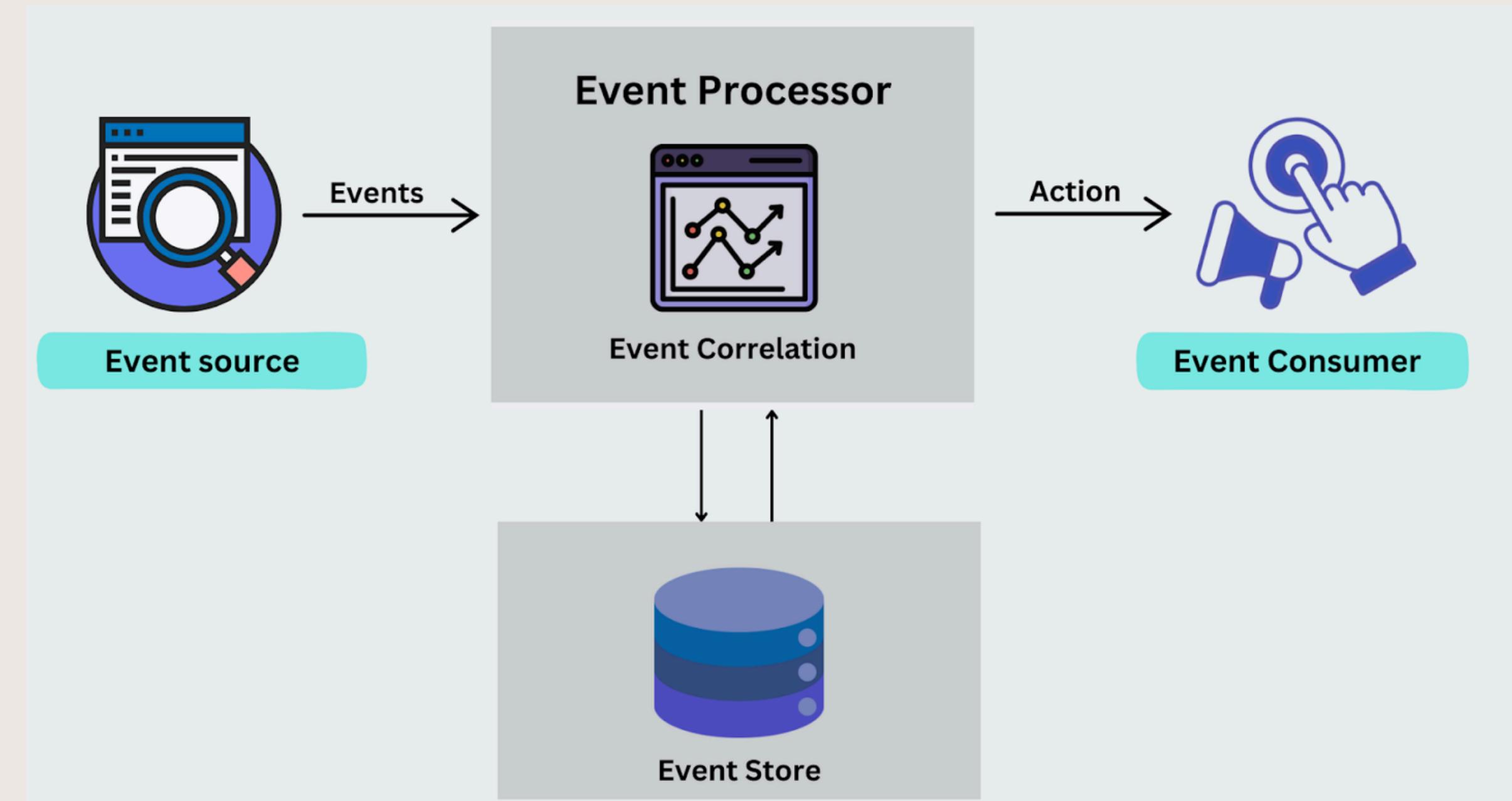
# Tipos de comunicación en EDA

- **Simple Event Processing:** Eventos procesados de forma individual.
- **Complex Event Processing:** Análisis de patrones y correlaciones entre eventos.
- **Event Stream Processing:** Flujo continuo de eventos en tiempo real.
- **Event Sourcing:** Almacenamiento histórico de todos los eventos del sistema.

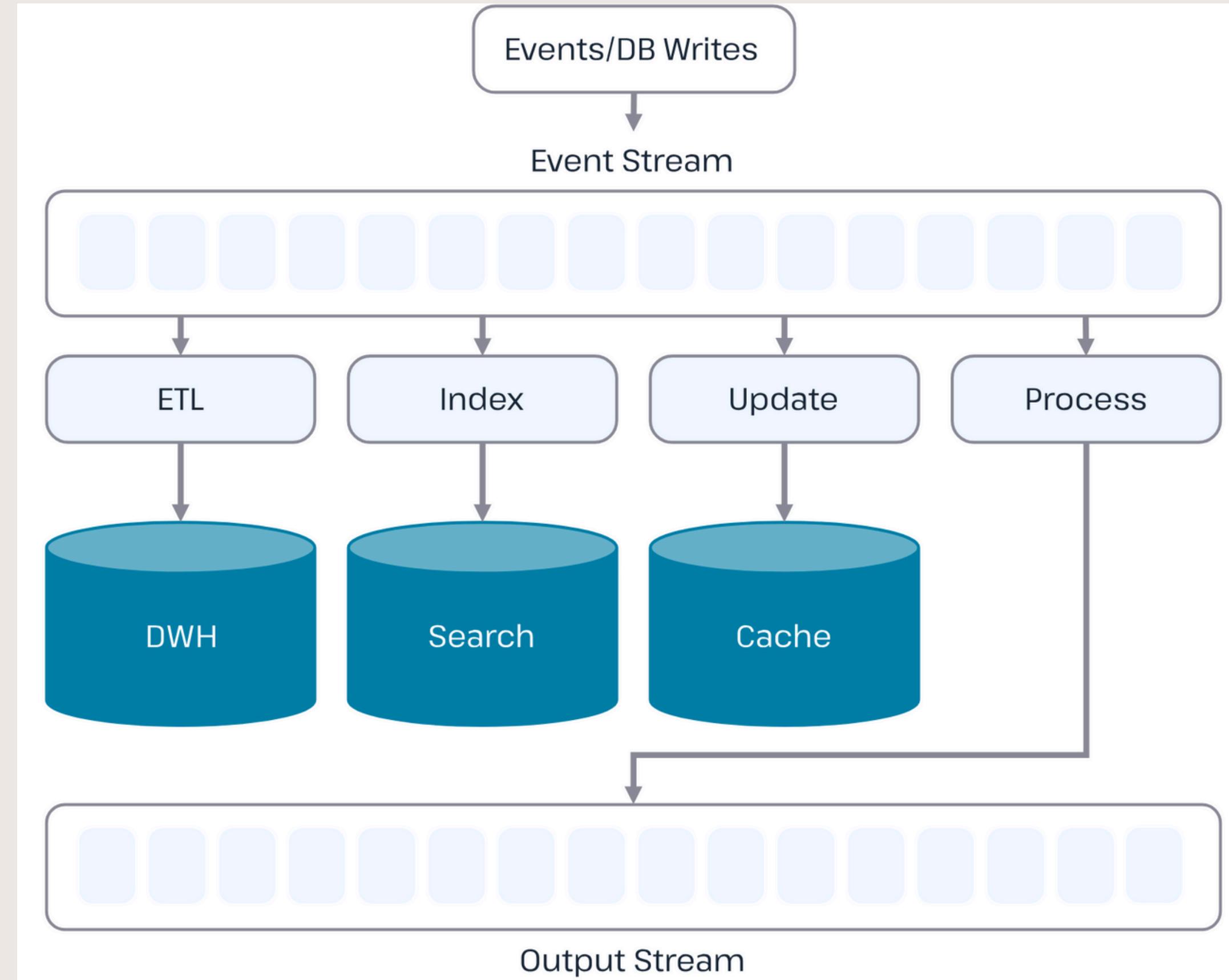
# Simple Event Processing



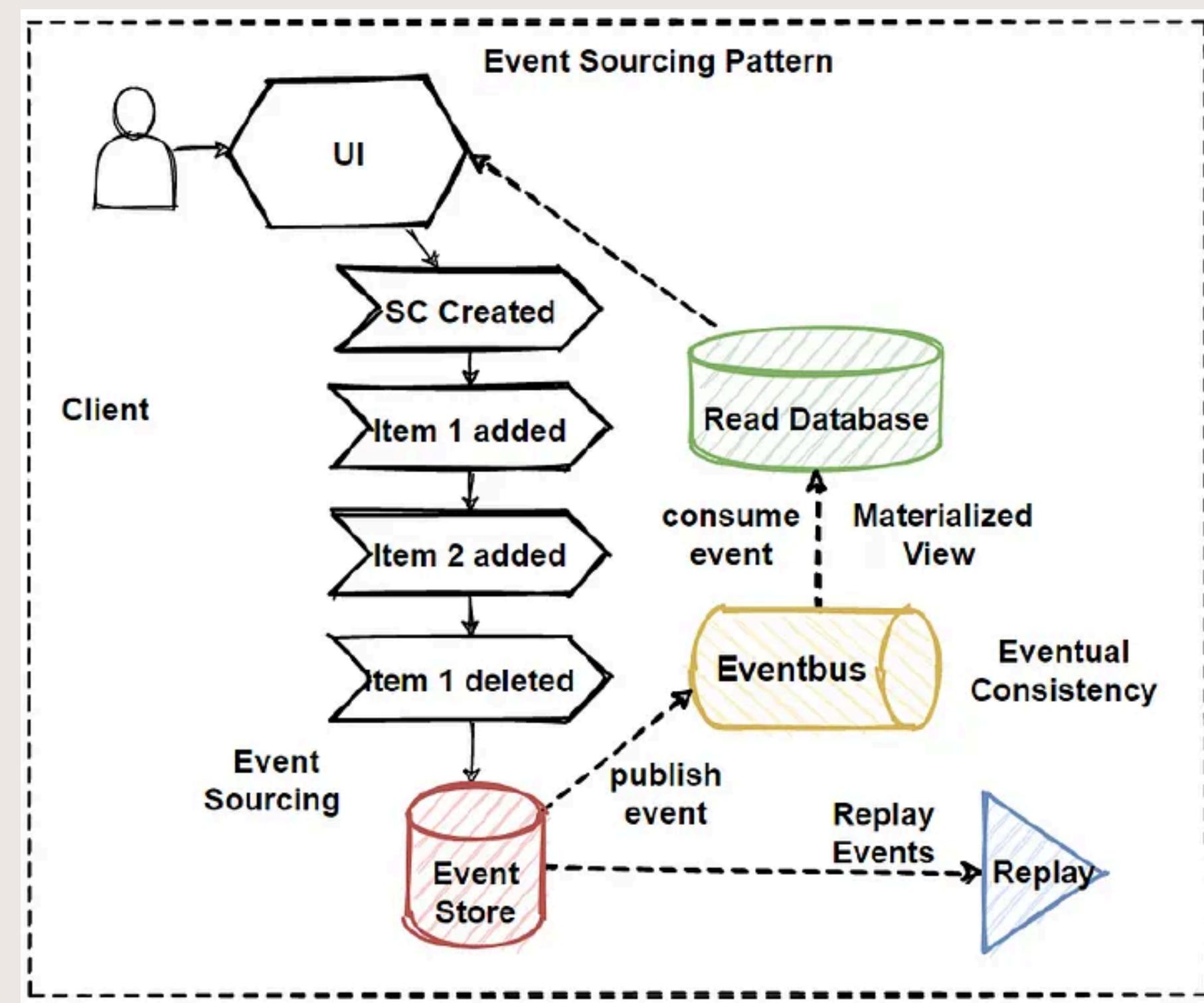
# Complex Event Processing



# Event Stream Processing



# Event Sourcing

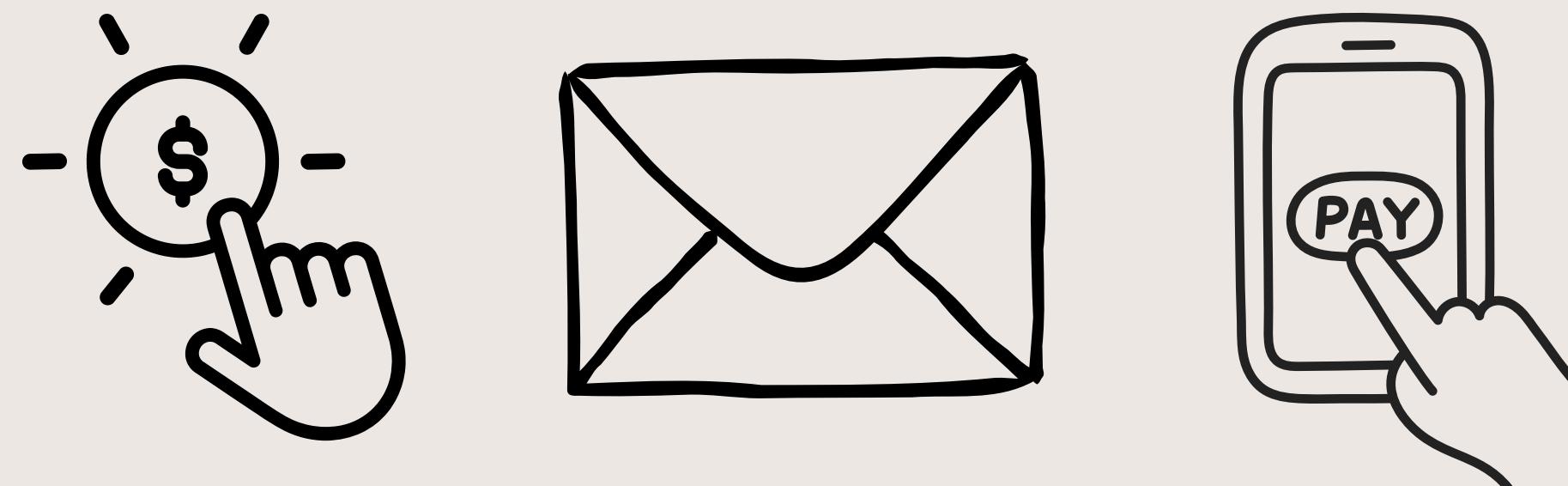


# Ventajas y Desventajas de EDA

VENTAJAS	DESVENTAJAS
Alta escalabilidad y flexibilidad	Complejidad en el diseño y orquestación de eventos
Desacopla componentes y facilita mantenimiento	Consistencia eventual, los datos no se actualizan simultáneamente
Respuesta en tiempo real ante eventos	Requiere monitoreo constante del flujo de eventos
Ideal para entornos distribuidos y microservicios	Mayor dificultad para depurar y trazar errores
Mejor resiliencia ante fallos	Sobrecarga en la gestión del broker de eventos
Permite integración sencilla con servicios serverless	

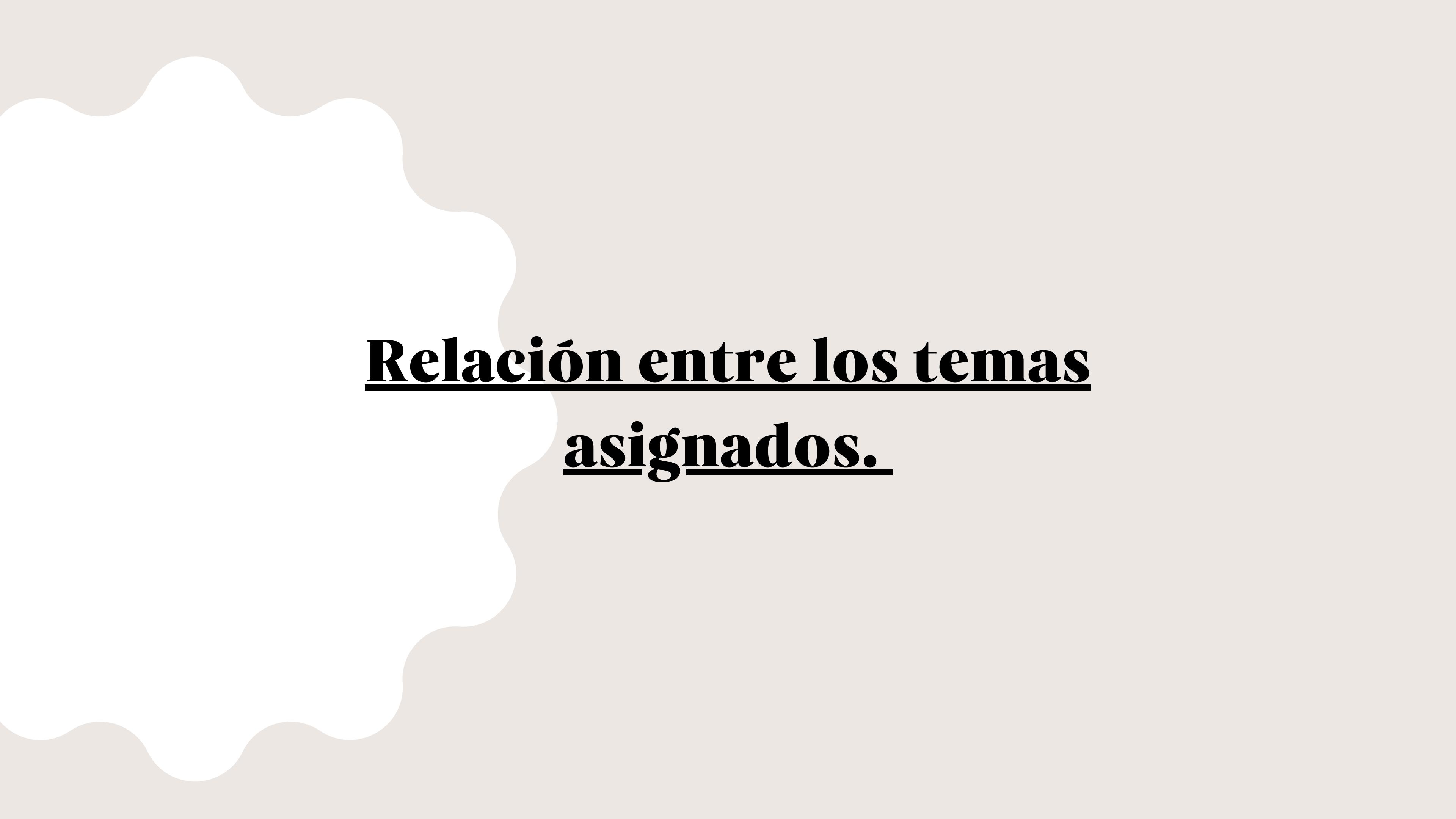
# Casos de uso de EDA

- Procesamiento de pagos en tiempo real
- Detección de fraudes financieros
- Sistemas IoT: monitoreo de sensores y dispositivos
- Analítica de comportamiento de usuarios
- Plataformas de streaming de datos
- Notificaciones y alertas distribuidas
- Orquestación de microservicios y workflows



# Casos de aplicación en la industria

	<b>Empresa</b>	<b>Implementación</b>	<b>Propósito</b>
	<b>Netflix</b>	Kafka	Procesa billones de eventos diarios para personalización.
	<b>Uber</b>	Kafka y Redis Streams	Coordinar viajes, pagos y ubicación.
	<b>Spotify</b>	Google Pub/Sub	Analiza interacciones y preferencias de usuarios.
	<b>Amazon</b>	SQS/SNS y Lambda	Actualiza inventarios y dispara flujos automáticos.

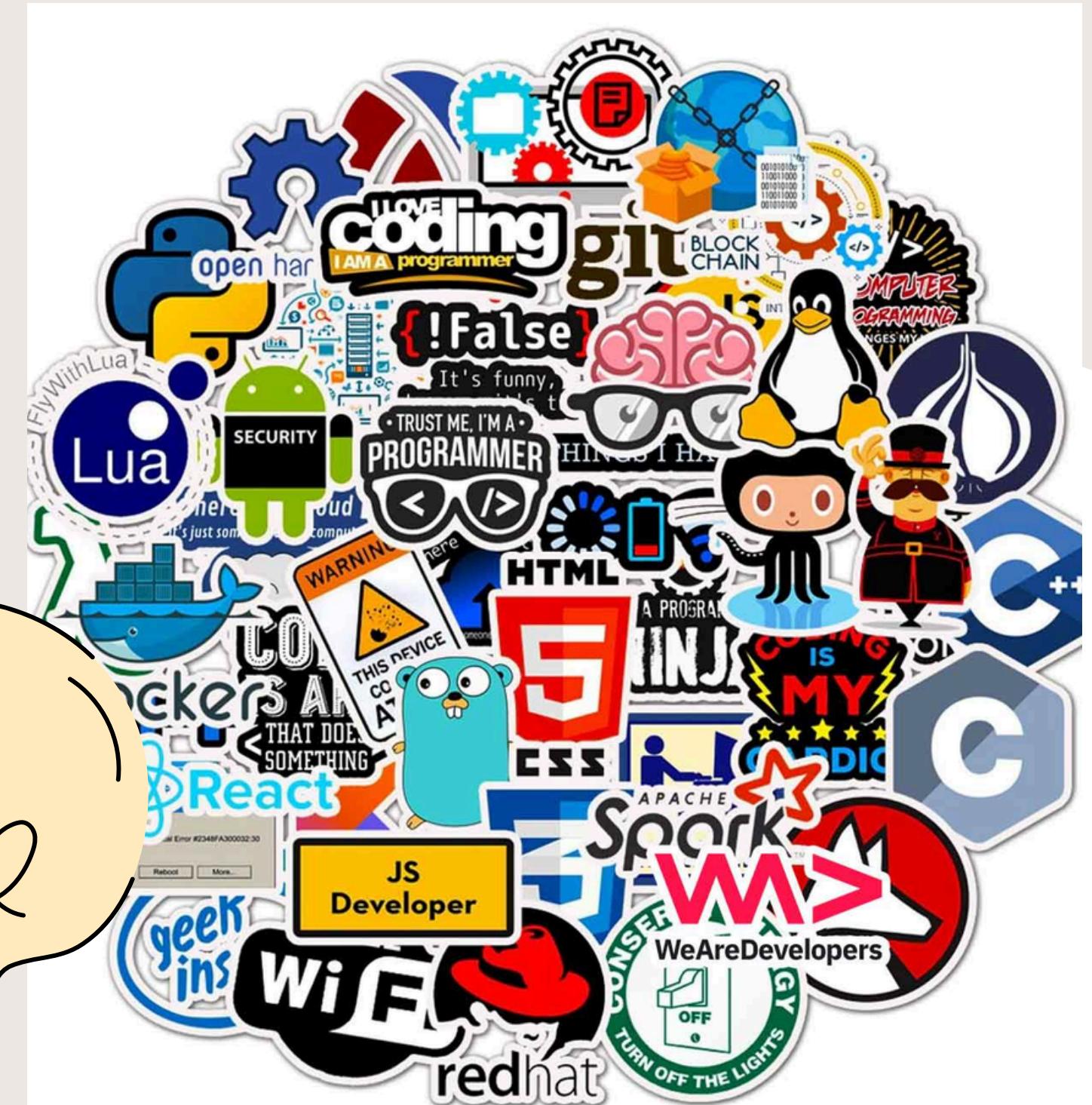
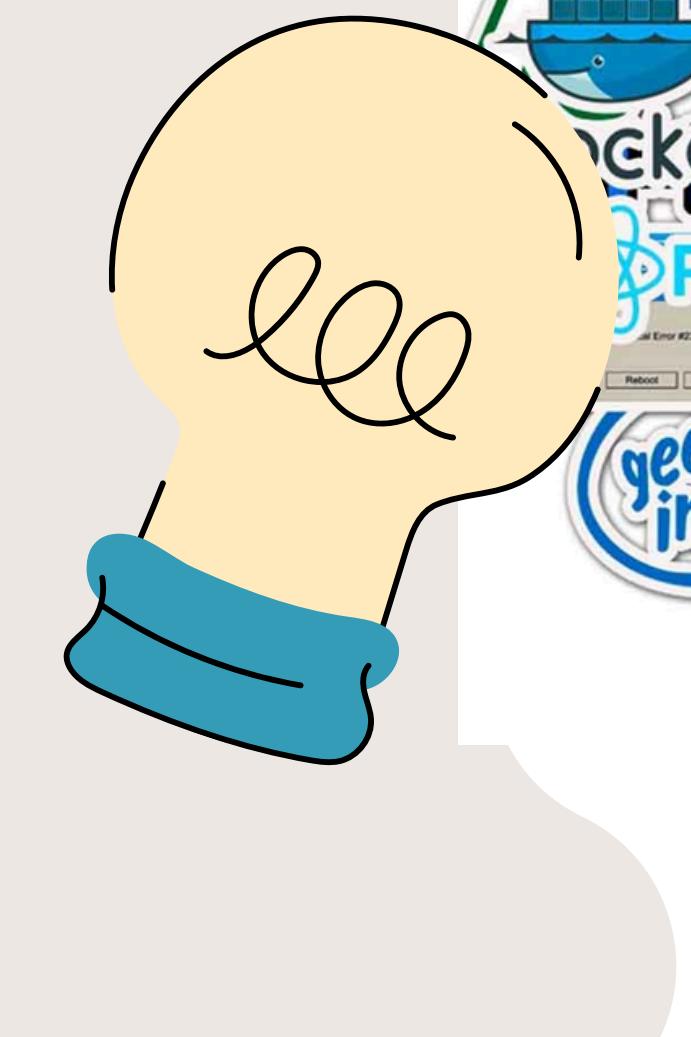


# **Relación entre los temas asignados.**



# Qué tan común es el stack

El stack compuesto por Astro, Ruby on Rails, Kafka y Cassandra es una combinación sólida y moderna. Aunque no es el más común en todas las empresas, se utiliza ampliamente en organizaciones que requieren procesamiento en tiempo real y resiliencia a gran escala. Kafka y Cassandra son herramientas estándar en el manejo de datos distribuidos, mientras que Rails permite un desarrollo rápido del backend y Astro garantiza alto rendimiento en el frontend. Este conjunto representa un ecosistema event-driven eficiente y adaptable a múltiples contextos industriales.



# Principios SOLID vs EDA

Principio	Nivel de Aplicación	Justificación
<b>S – Single Responsibility</b>	Alto	componentes por evento con responsabilidad única.
<b>O – Open/Closed</b>	Media	Sistemas extensibles por nuevos consumidores sin cambiar productores.
<b>L – Liskov Substitution</b>	Bajo	no aplica directamente al nivel de arquitectura.
<b>I – Interface Segregation</b>	Alto	consumidores se suscriben sólo a interfaces (eventos) que necesitan.
<b>D – Dependency Inversion</b>	Media-Alta	Desacoplamiento mediante eventos favorece invertir dependencias en runtime.

# Atributos de calidad vs EDA

Aspecto	Descripción
Rendimiento	Variable: depende de diseño y latencia de eventos.
Escalabilidad	Alta: diseñado para escalar horizontalmente.
Disponibilidad	Alta: desacoplamiento y replicación aumentan disponibilidad.
Consistencia	Eventual consistency común.
Mantenibilidad	Media: más servicios = más complejidad.

# Analisis de tacticas vs EDA

Aspecto	Descripción
Caching	Sí: caches de eventos o materialized views.
Replicación	Sí: replicación de eventos entre servicios.
Particionado / Sharding	Sí para tópicos/eventos y carga.
Backpressure	Necesario en flujos intensos.
Event Sourcing / CQRS	Naturalmente compatible.

# Analisis de patrones vs EDA

Aspecto	Descripción
Pub/Sub	Los productores publican eventos sin conocer a los consumidores. Los consumidores se suscriben solo a los temas que les interesan, promoviendo flexibilidad y bajo acoplamiento.
Event Sourcing	Guarda cada cambio del sistema como un evento inmutable, lo que permite reconstruir estados, auditar operaciones y mejorar la trazabilidad.
Event Stream Processing	Procesa flujos continuos de eventos en tiempo real para analítica, monitoreo o respuesta automática.
CQRS	Separa las operaciones de lectura y escritura, optimizando el rendimiento en sistemas que procesan grandes volúmenes de eventos.
Producer-Consumer	Los productores generan eventos y los consumidores los procesan de forma asíncrona. Permite desacoplar el flujo de trabajo, balancear carga y escalar cada parte de manera independiente.

# Mercado laboral vs EDA

Aspecto	Descripción
Demanda	Alta: arquitectos y diseñadores EDA buscados en empresas de datos.
Roles	Arquitecto de software, ingeniero de integración, SRE.
Nivel de especialización	Alto: comprensión de concurrencia, consistencia y diseño distribuido.
Tecnologías destacadas	Apache Kafka, RabbitMQ, ActiveMQ, AWS SNS/SQS, Azure Service Bus, Google Pub/Sub, Redpanda, Pulsar.
Tendencia	Creciente, impulsada por el auge de arquitecturas event-driven, analítica en tiempo real y soluciones serverless.
Salario promedio LATAM	Entre 5 y 9 millones de pesos mensuales, dependiendo del rol y experiencia.



A astro

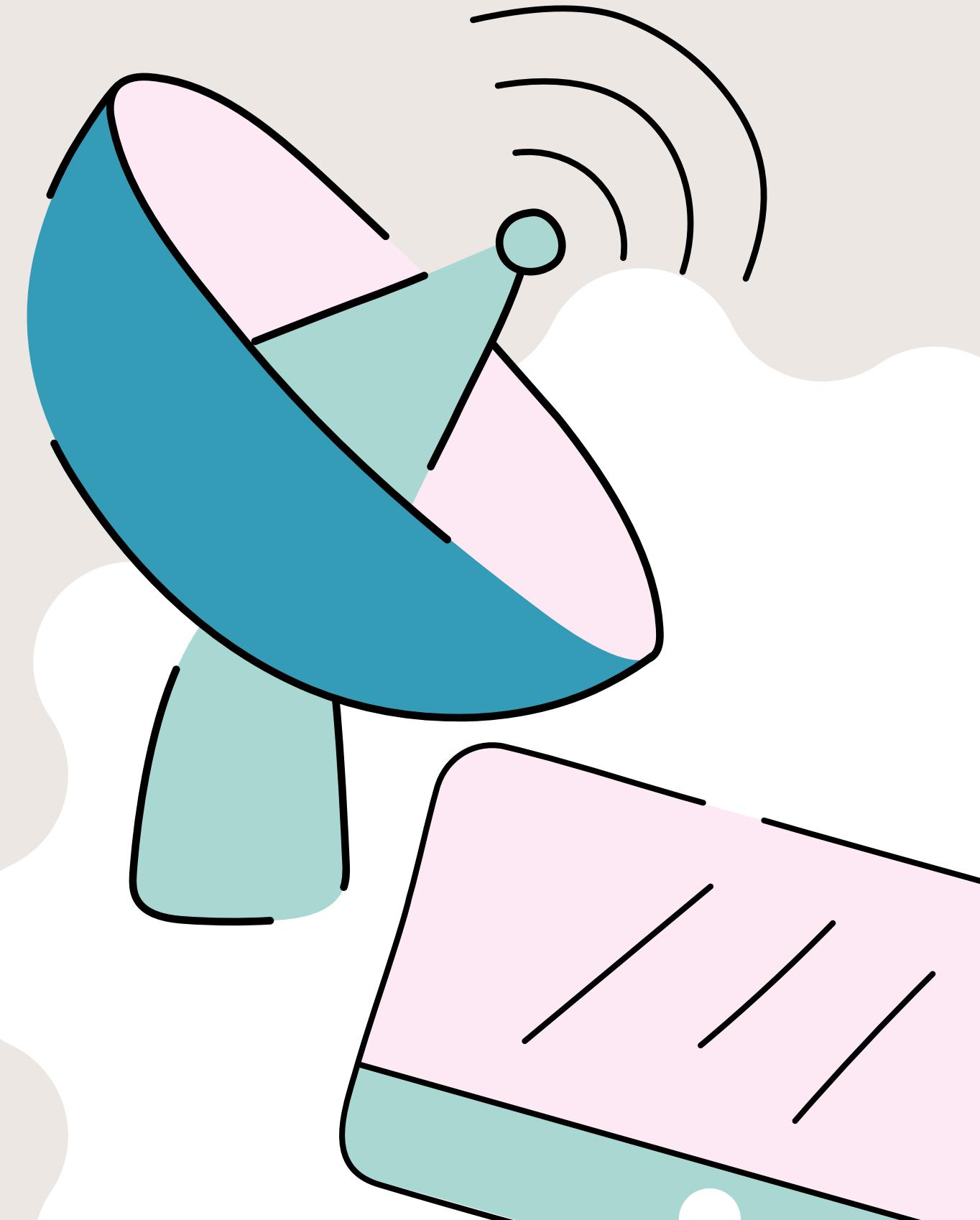
# Definicion

Astro es un framework para paginas web Open Source basado en JavaScript y escrito en Go y Typescript por Fred K. Schott.

# Características

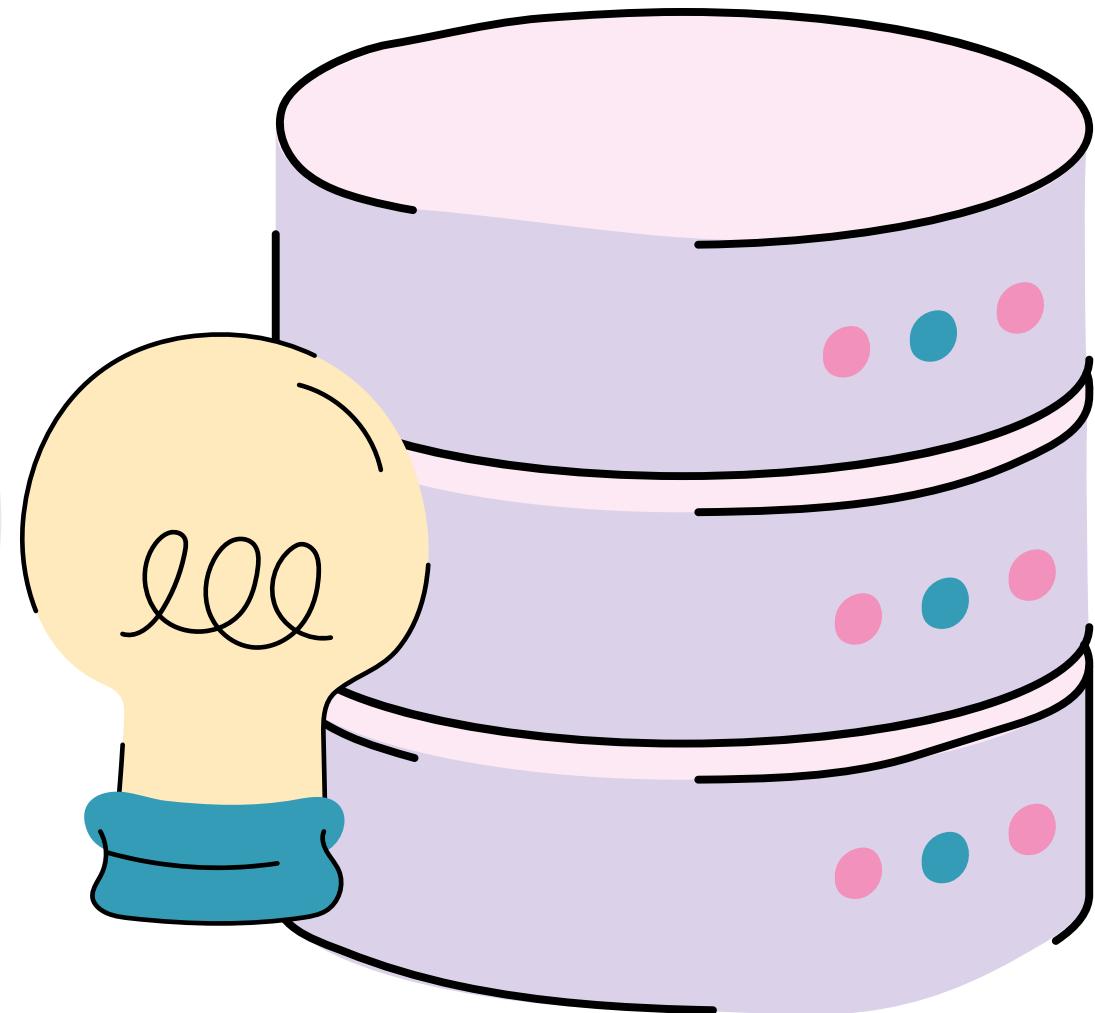
Astro es orientado a contenido, y es considerado el framework para páginas web más ligero y eficientes de usar gracias a la arquitectura creada por este conocida como Astro Islands.

Cuenta con soporte oficial para múltiples frameworks como React, Vue, Svelte, Cloudflare y Tailwind



# Historia y Evolución

- 2021: Beta
- 2022: Versión 1.0
- 2023: Versión 2.0 y 3.0
- 2024: Versión 4.0



# Pros

- Gran flexibilidad
- Eficiente
- Ligero
- Gran desempeño por medio de renderización de componentes en servidor

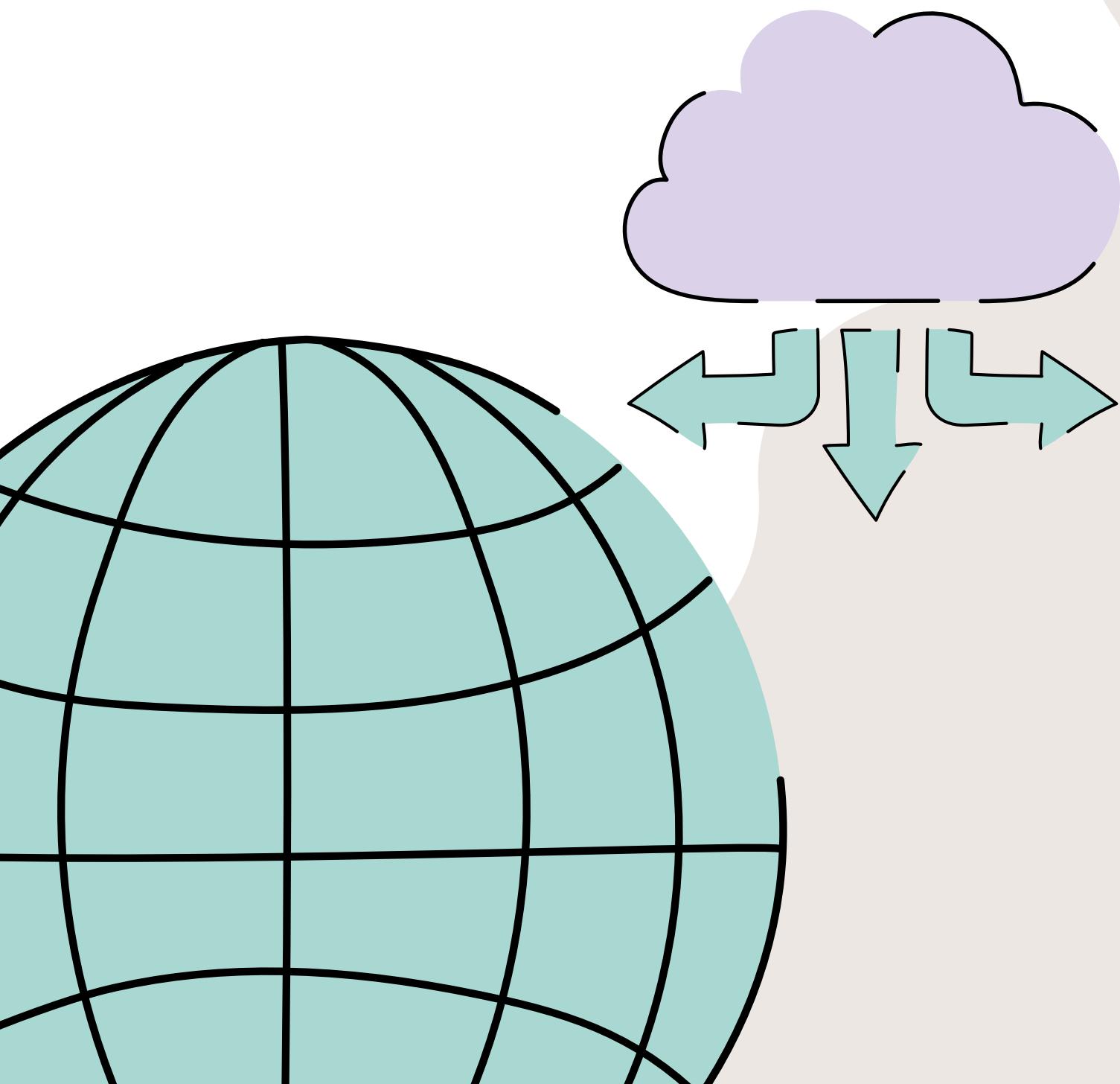
# Contras

- No apto para aplicaciones dinámicas o masivas como redes sociales

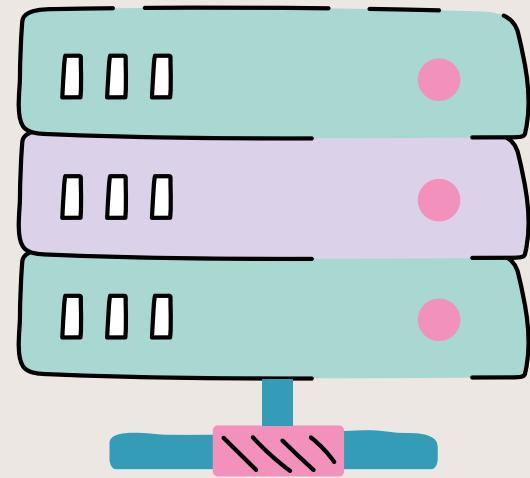
**versus**



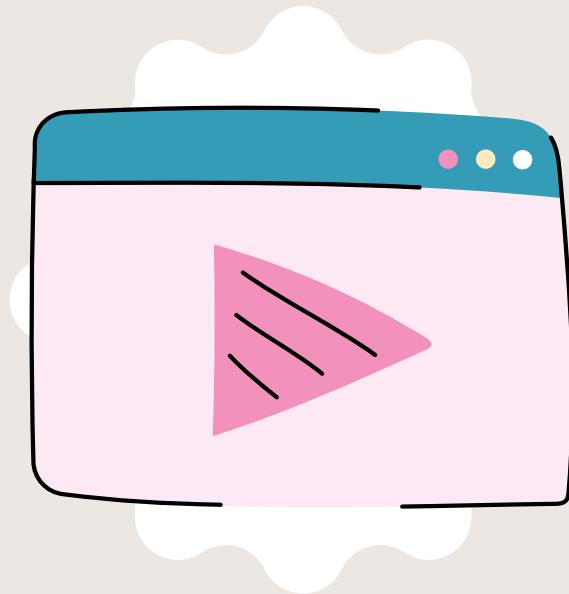
# Casos de uso



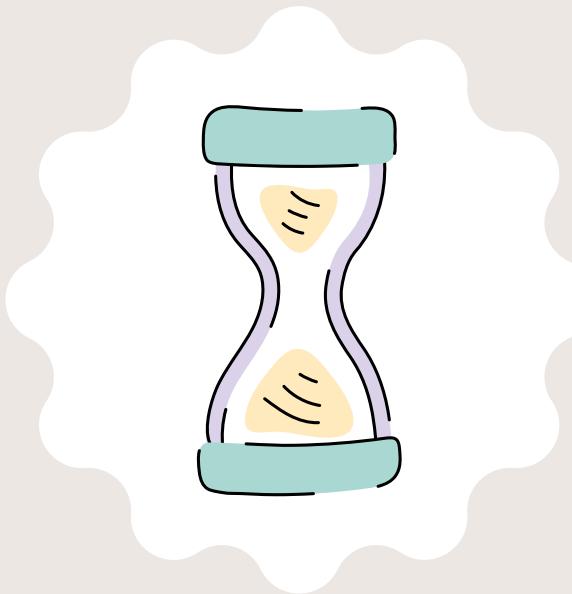
Astro es un excelente framework para páginas web estáticas orientadas a contenido como foros o blogs



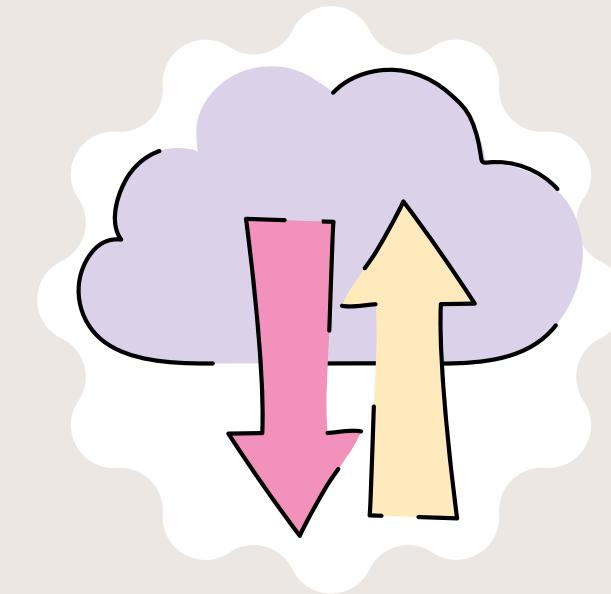
# Casos de Aplicación en la Industria



**Porsche**



**The Guardian**



**Visa**

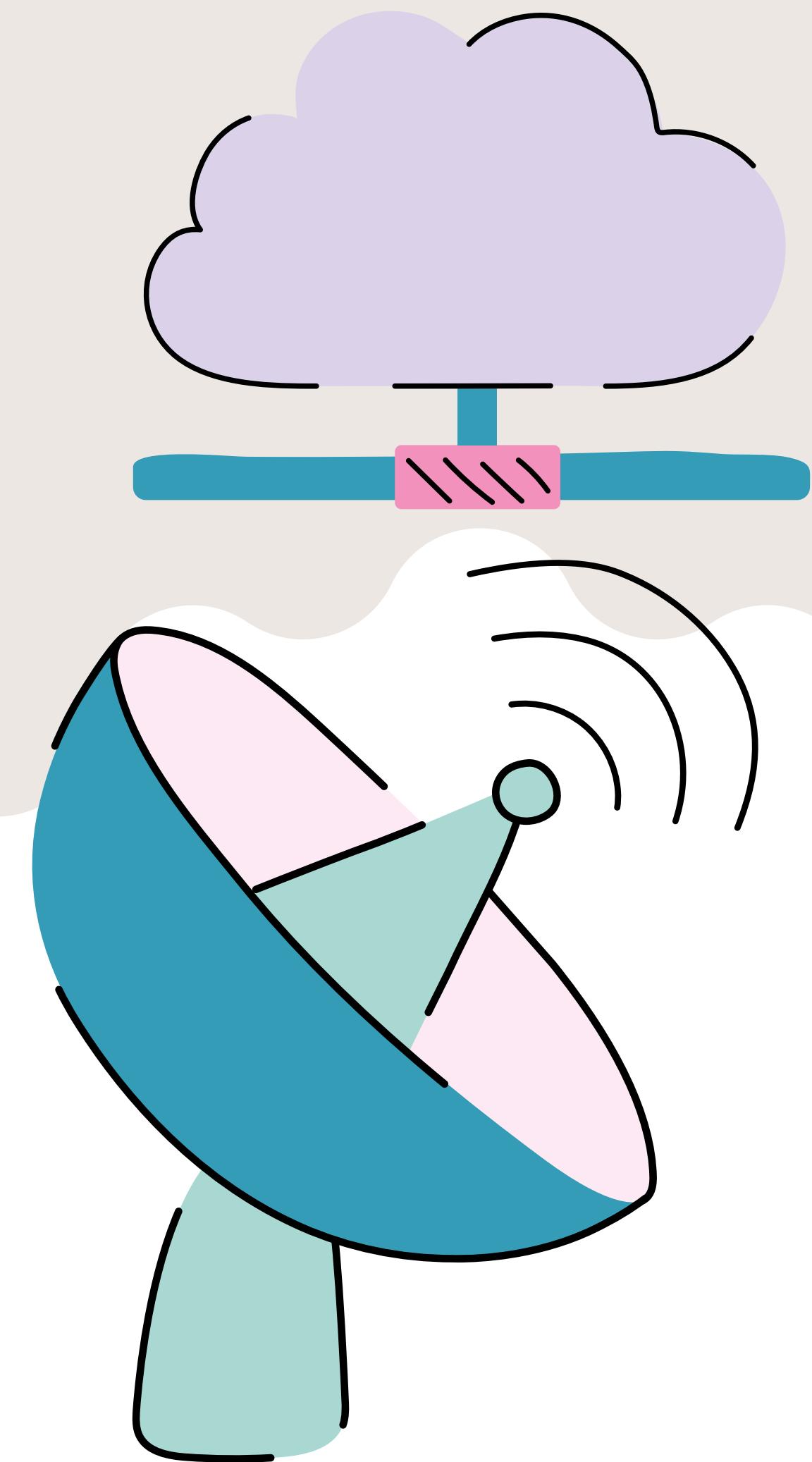


# Definicion

Ruby es un lenguaje de uso general inspirado en Perl, Lisp, BASIC, entre otros.

El objetivo principal de su creador es que el programador sea productivo y se divierta.

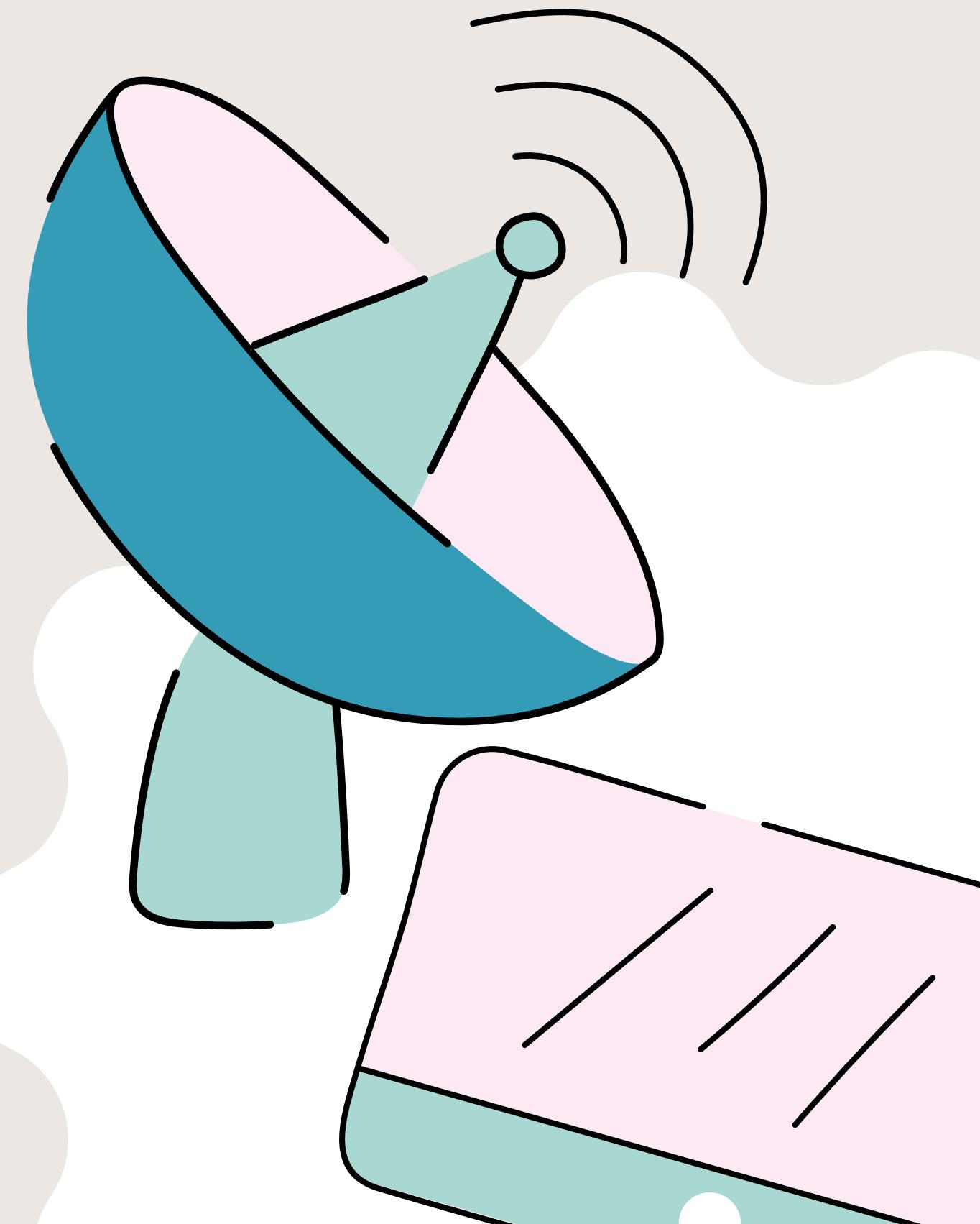
Es muy similar a Python tanto en funcionalidades como su facilidad de aprender



# Características

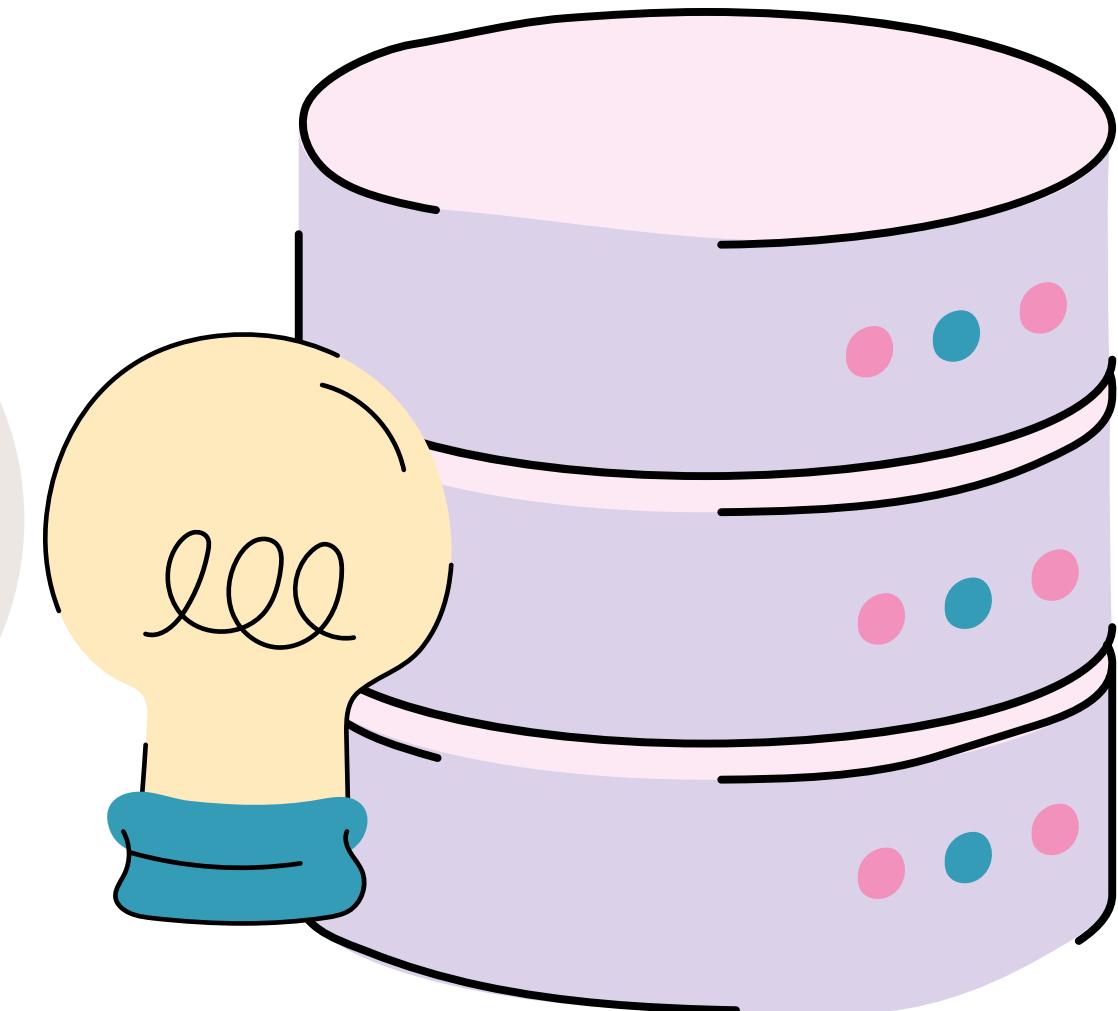
Soporta paradigmas de objetos, funcional y procesal (procedural).

Su syntax es similar a Perl, mientras que la semántica es similar a Smalltalk, pero la filosofía detrás de este es muy diferente a la de Python



# Historia y Evolución

- 1993: Yukihiro Matsumoto empieza a desarrollar Ruby
- 1995: Primera versión estable de Ruby es disponible en Japon
- 2000: Ruby supera en popularidad a Python en Japon
- 2003: Ruby 1.8
- 2013: Ruby 2.0, Ruby 1.8 es deprecado
- 2020: Ruby 3, también conocido como Ruby 3x3, ya que los programas son 3 veces más rápidos que con Ruby 2
- Diciembre 2024: Ultima versión estable de Ruby (Ruby 3.4)



# Pros

- Múltiples librerías (Gems)
- Bajo costo de utilizar
- Fácil de aprender y de leer

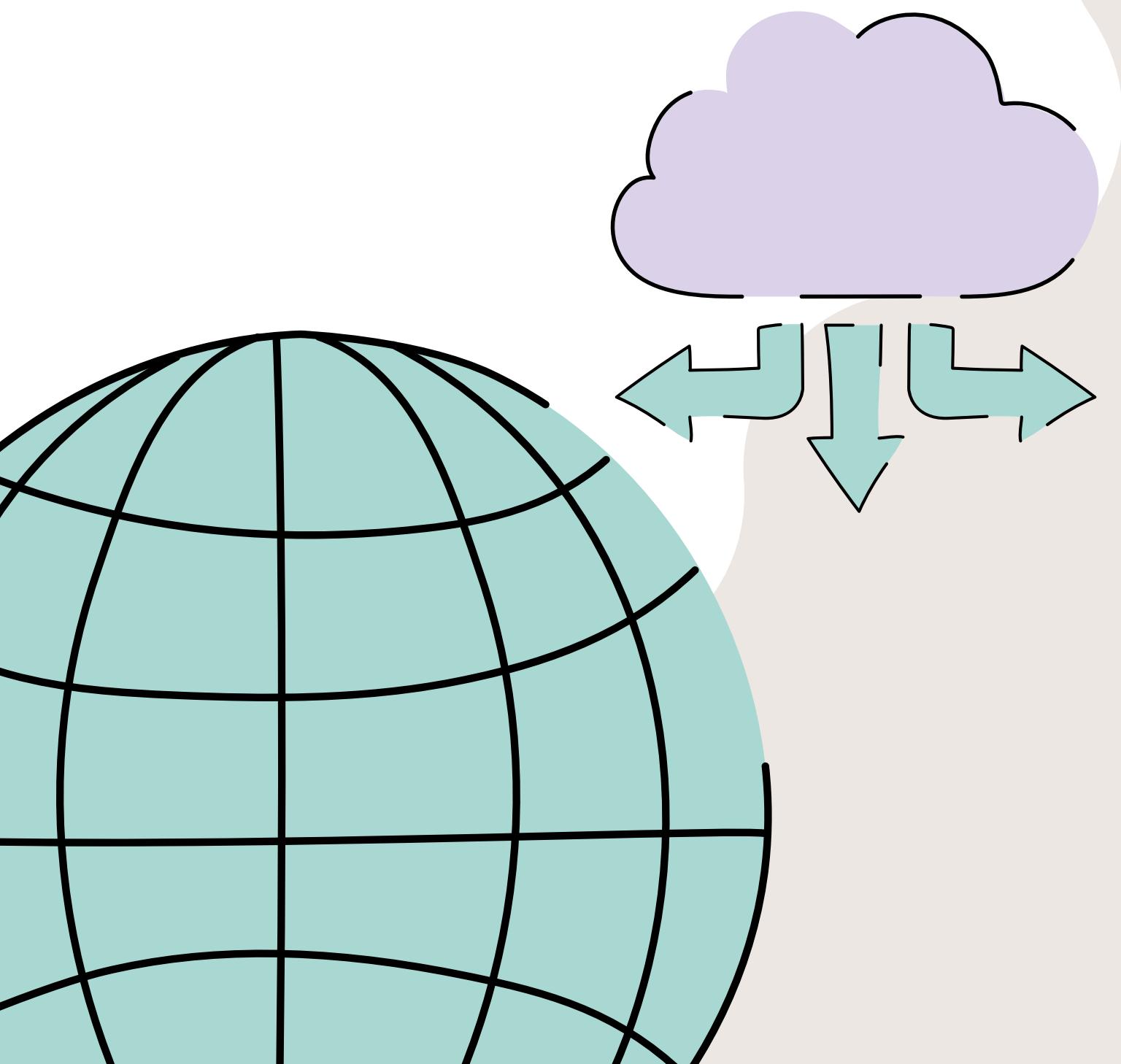
# Contras

- Poca flexibilidad
- Mala documentación

**versus**

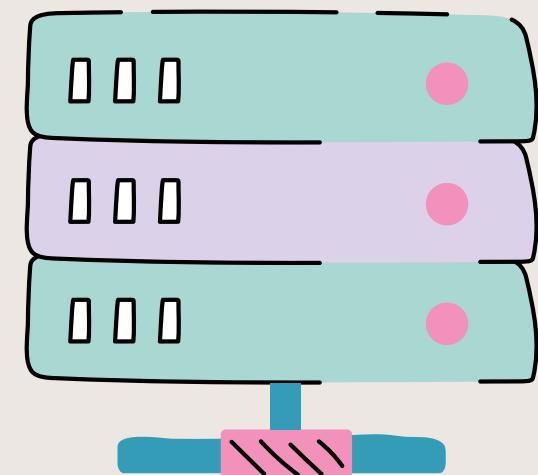


# Casos de uso



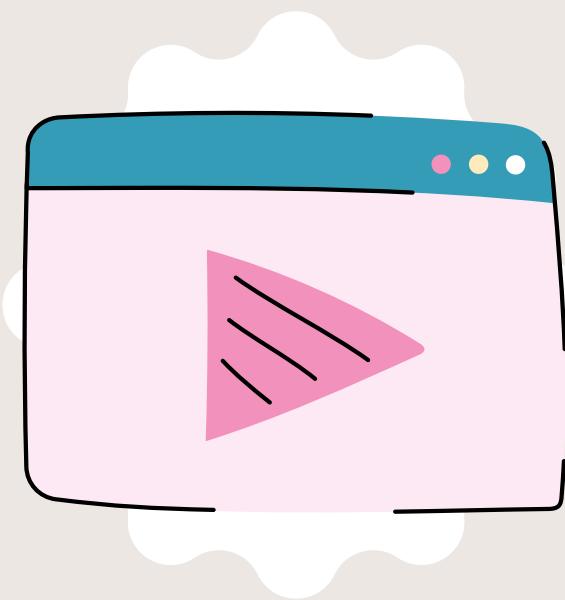
En general, Ruby puede sustituir a Python en cualquier caso de desarrollo Backend. Por ende, sus casos de uso más comunes incluyen:

- Desarrollo Web
- Sistemas electrónicos
- Automatización
- e-Commerce

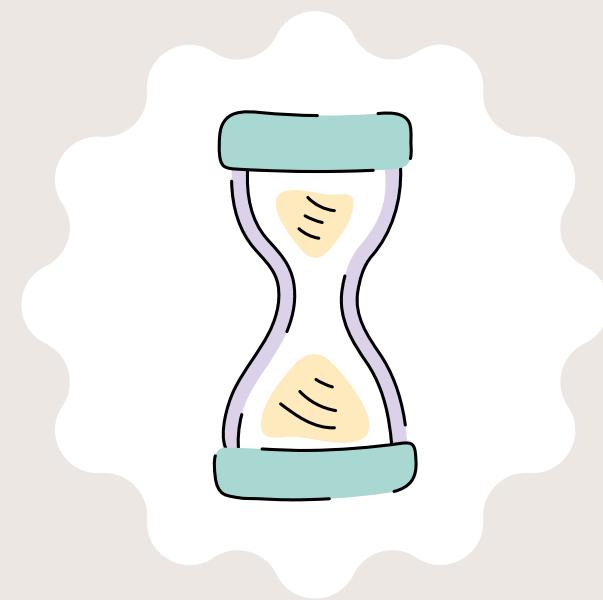


# Casos de Aplicación en la Industria

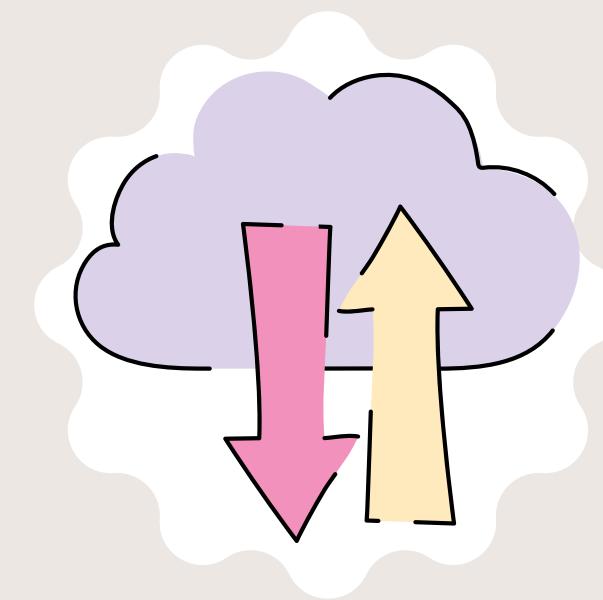
Los principales usos de Ruby ocurren en desarrollo Web gracias a Ruby on Rails



**GitHub**



**Shopify**



**AirBnB**



# Definicion

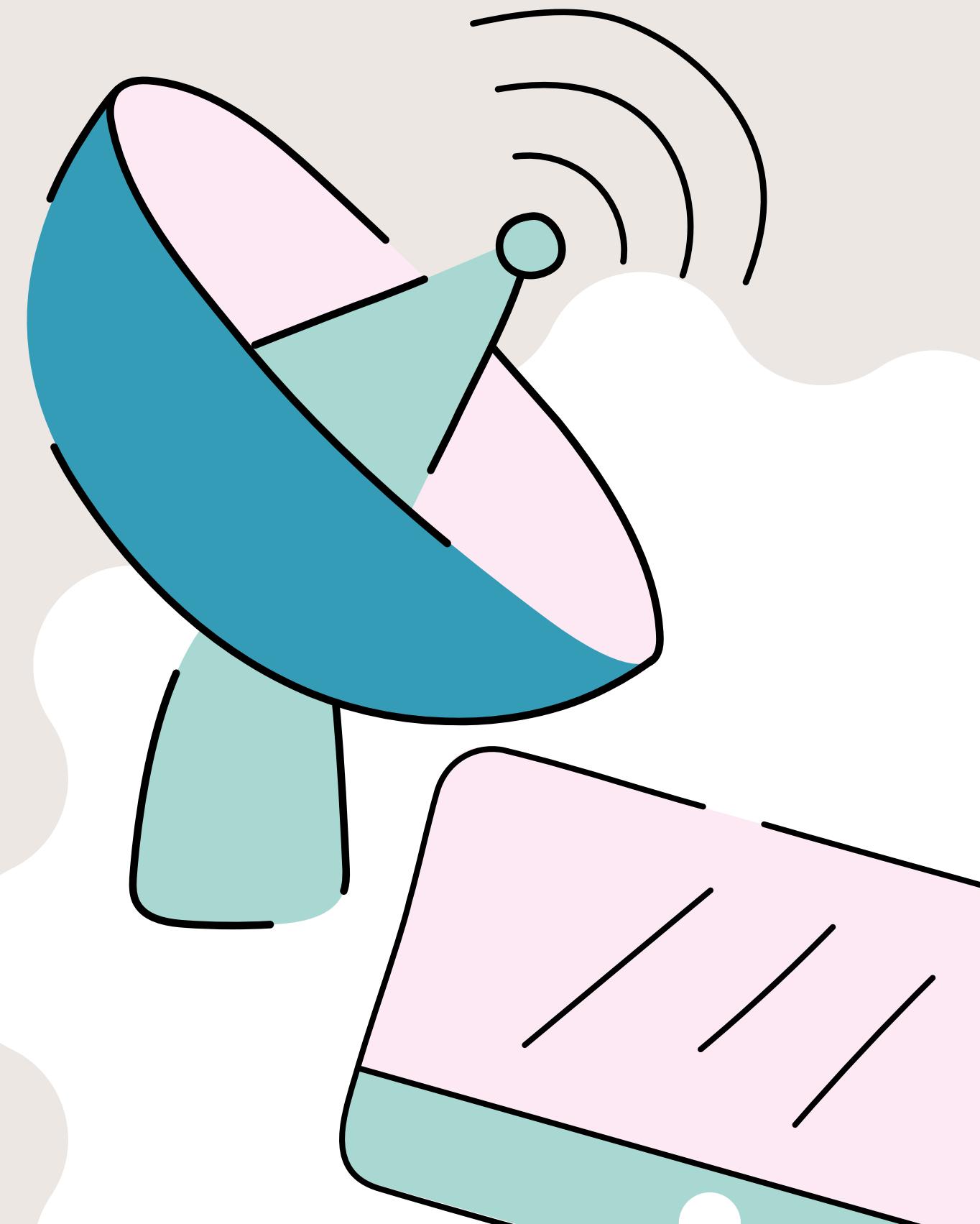
Rails es un framework MVC para aplicaciones web para Ruby

Este framework ha influenciado una gran cantidad de frameworks para aplicaciones web para otros lenguajes, incluyendo Django, Grails, CakePHP y Sails.js

# Características

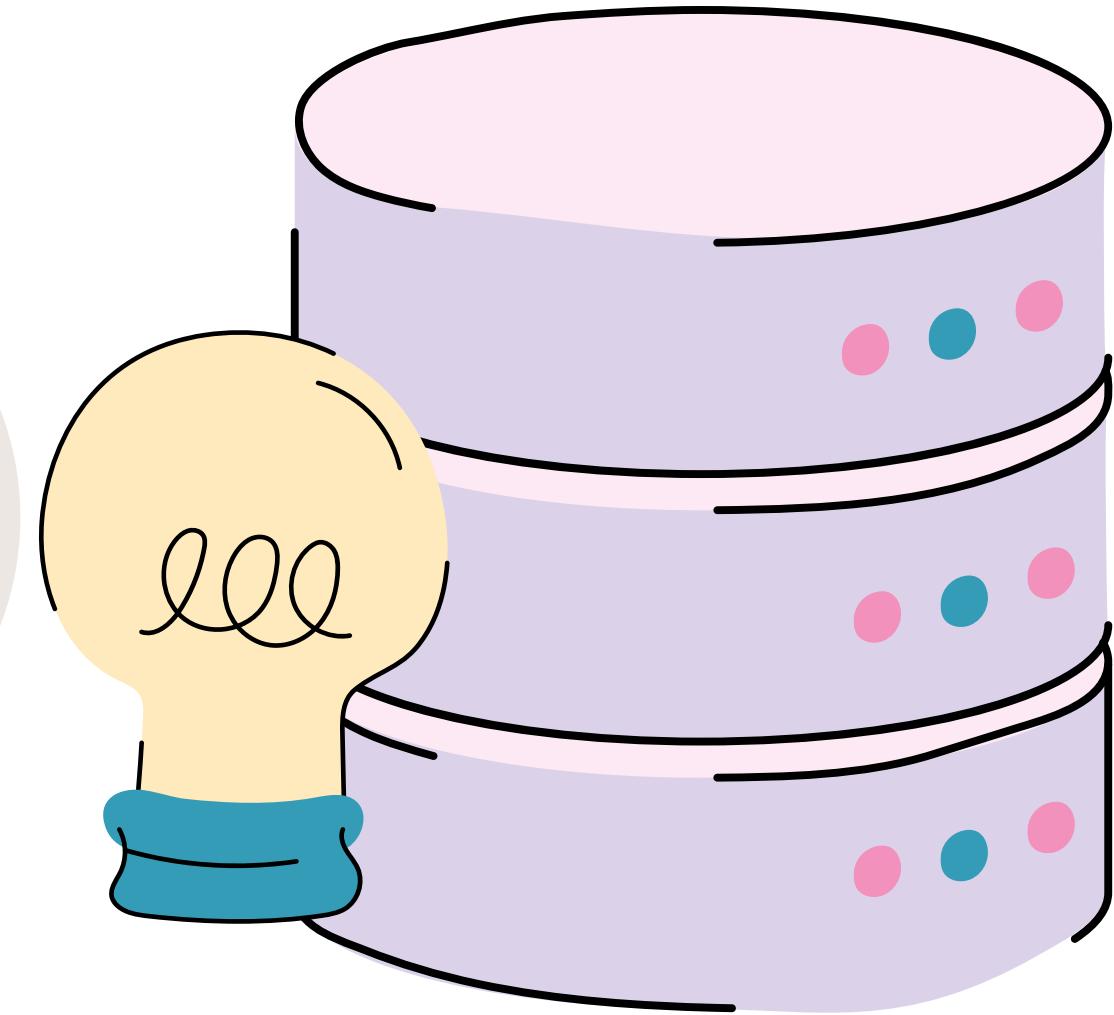
Rails facilita el uso de estándares web como XML o Json para data y HTML, CSS y Javascript para interfaces, ademas de implementar paradigmas como CoC y DRY.

Rails es famoso por su integración con base de datos para creación y migracion



# Historia y Evolución

- 2004: David Heinemeier Hansson publica la primera versión de Rails
- 2007: Apple incluye Rails en Mac OSx Leopard
- 2009: Rails 2.3
- 2011: Rails 3.1, esta versión incluye jQuery como la librería de JavaScript por defecto
- 2013: Rails 4.0
- 2016: Rails 5.0
- 2019: Rails 6.0
- 2021: Rails 7.0
- 2024: Rails 8.0

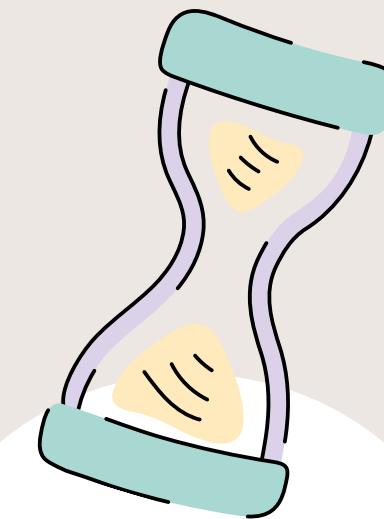


# Pros

- Seguridad
- Simple de usar
- Fácil implementación de lógica de negocios
- Compatible con otros frameworks

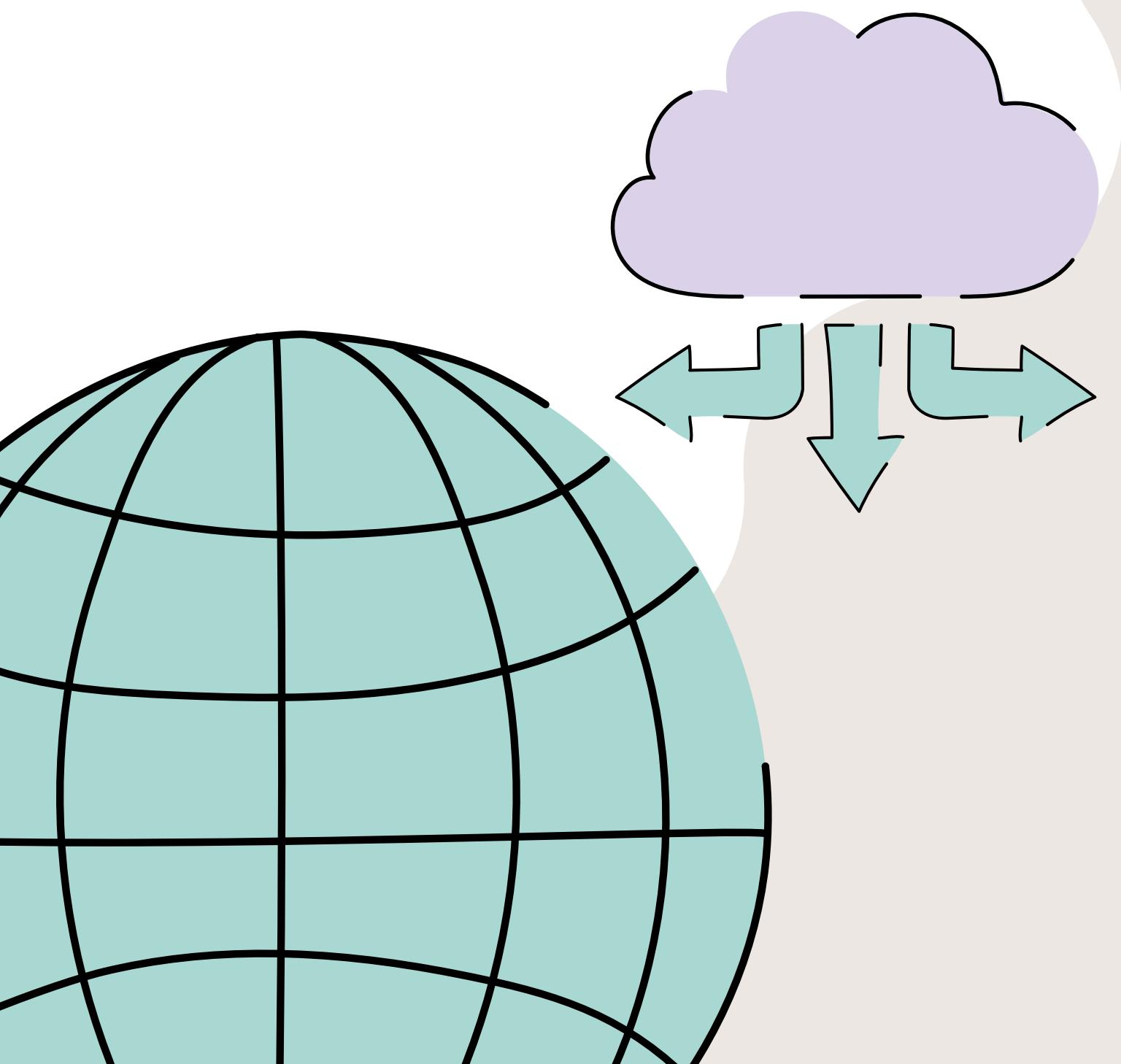
# Contras

- Poco flexible
- No es tan popular como otros frameworks

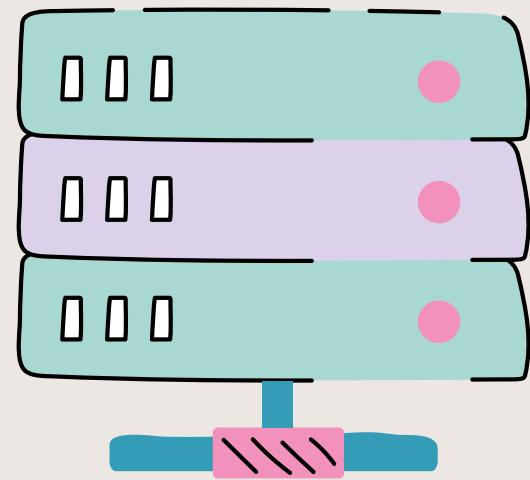


**versus**

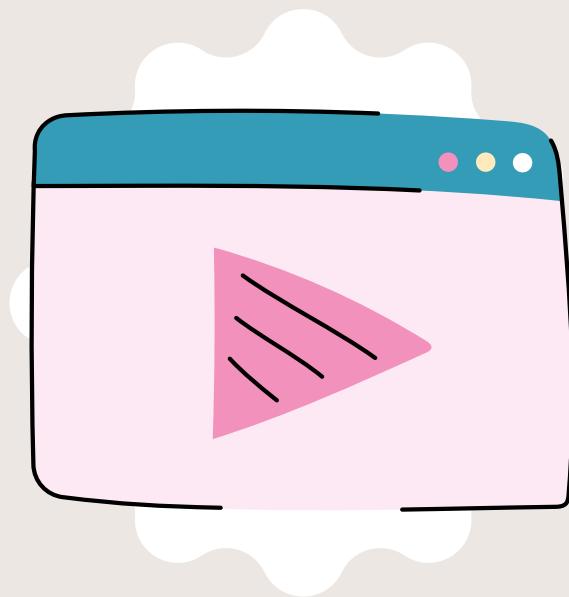
# Casos de uso



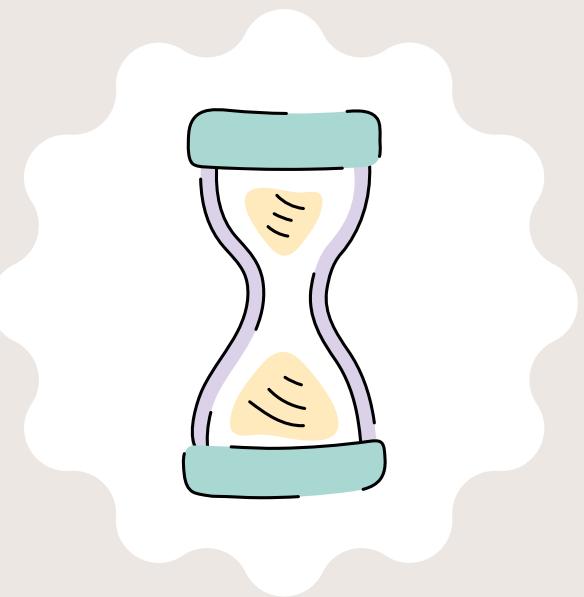
- E-commerce
- APIs
- Redes sociales
- Dashboards



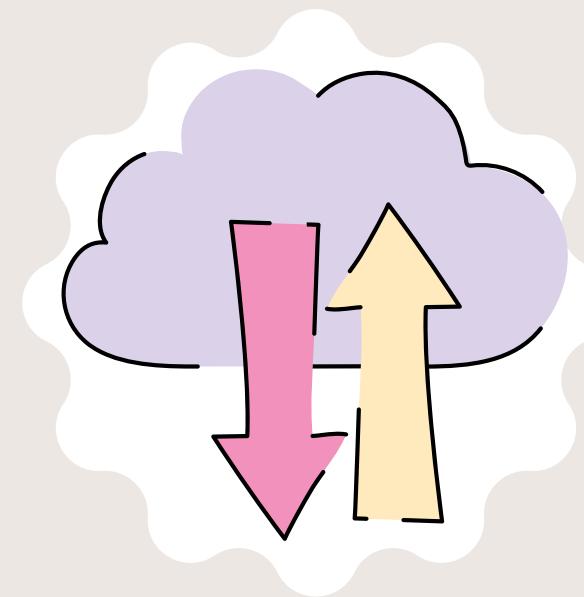
# Casos de Aplicación en la Industria



**Soundcloud**



**Indiegogo**



**Hulu**



**kafka**

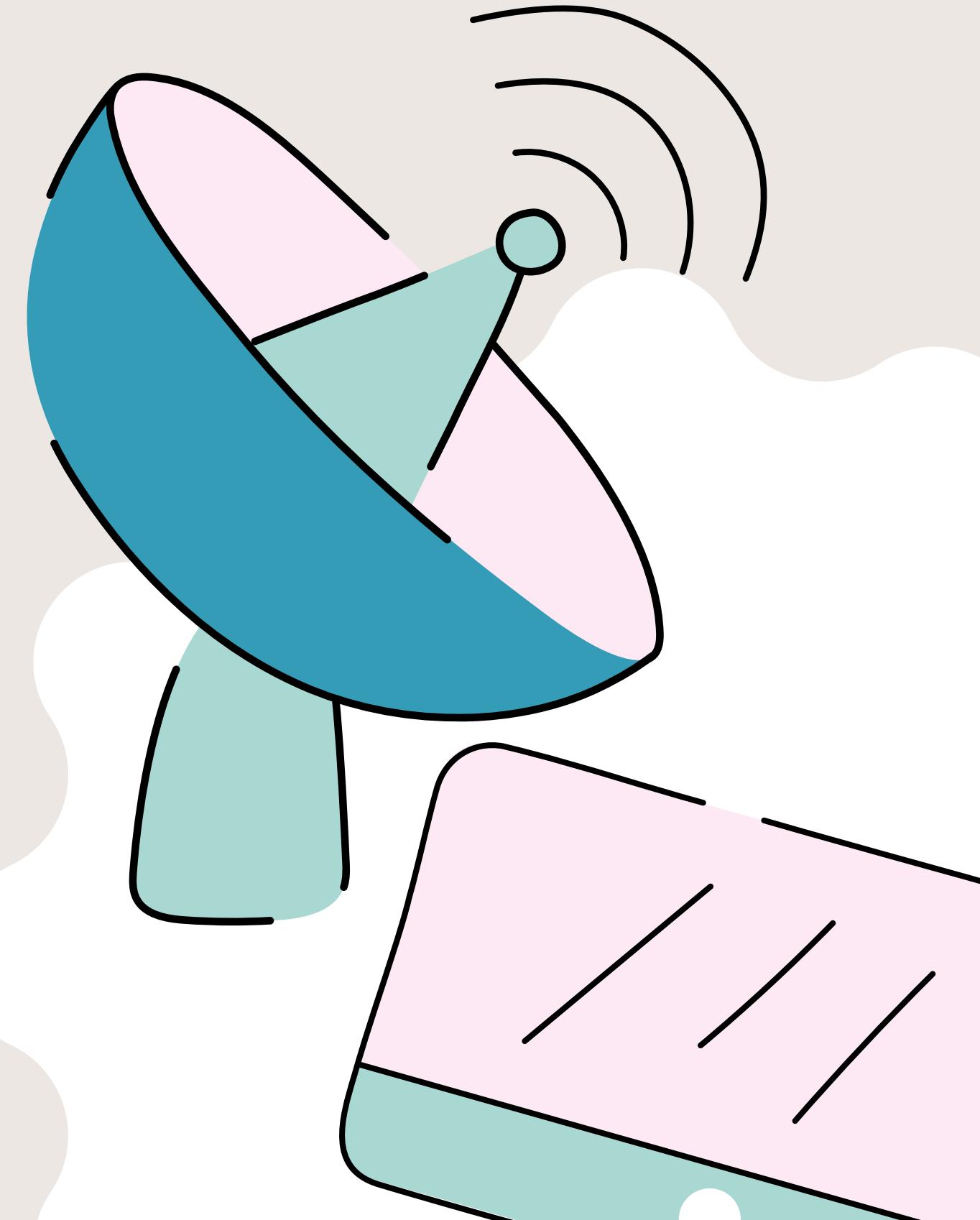
# Definicion

Apache Kafka es una plataforma open source de envío de eventos escrita en Java y Scala, desarrollada en LinkedIn por Jay Kreps, Neha Narkhede y Jun Rao en el 2010.

Su nombre viene del autor Franz Kafka, Kreps escogió este nombre debido a que es fan del autor

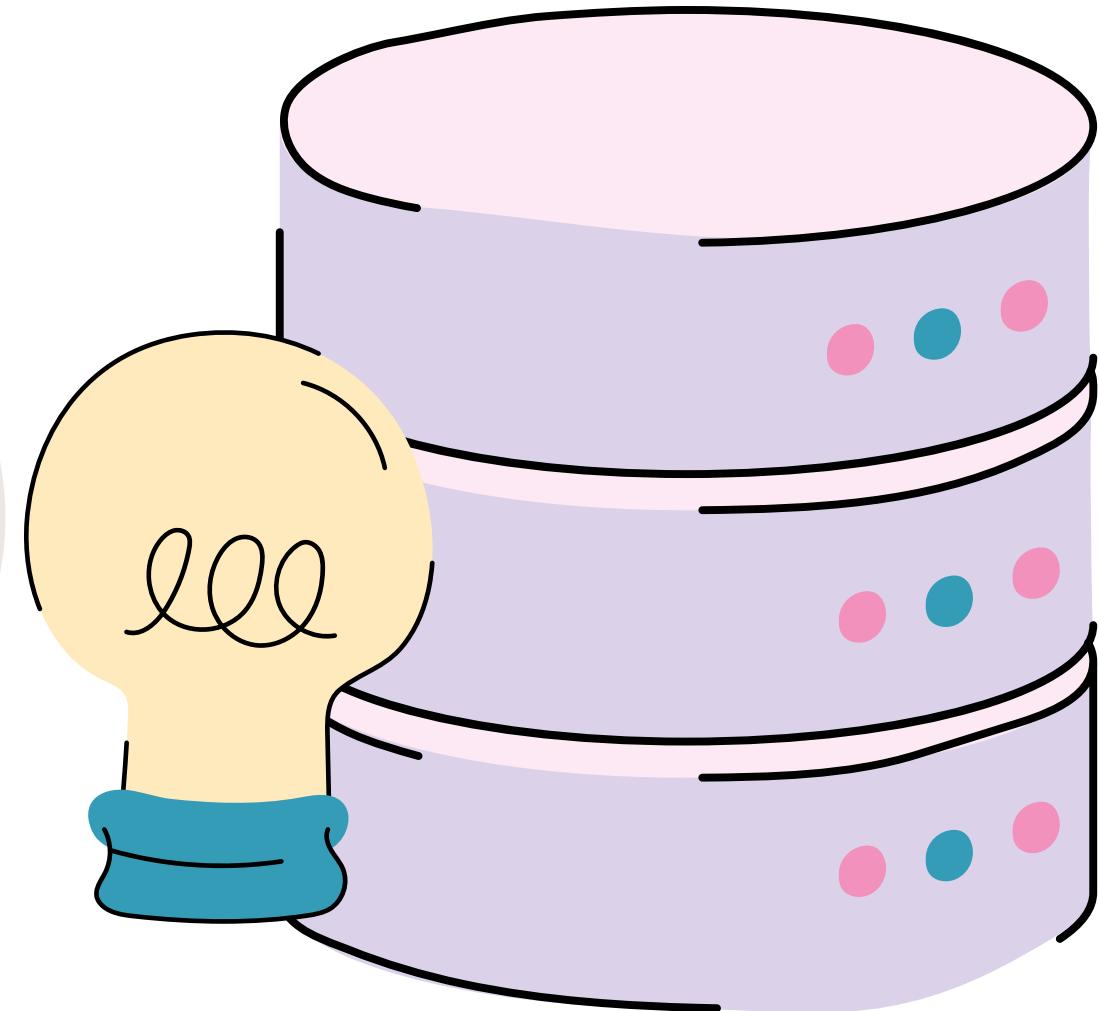
# Características

- Alto throughput: Latencias tan bajas como 2ms
- Escalabilidad: Capacidad para miles de brokers, trillones de mensajes diarios y pentabytes de data
- Almacenaje permanente: Clusters distribuidos tolerantes a fallas
- Alta disponibilidad



# Historia y Evolución

- 2010: Jay Kreps, Neha Narkhede y Jun Rao empiezan el desarrollo de Kafka en LinkedIn
- 2011: LinkedIn implementa Kafka en sus operaciones
- 2012: Kafka se vuelve Open Source y se une a Apache Software Foundation
- 2025: Ultima versión estable



# Pros

- Fácil integración
- Alto desempeño
- Seguridad
- Documentacion

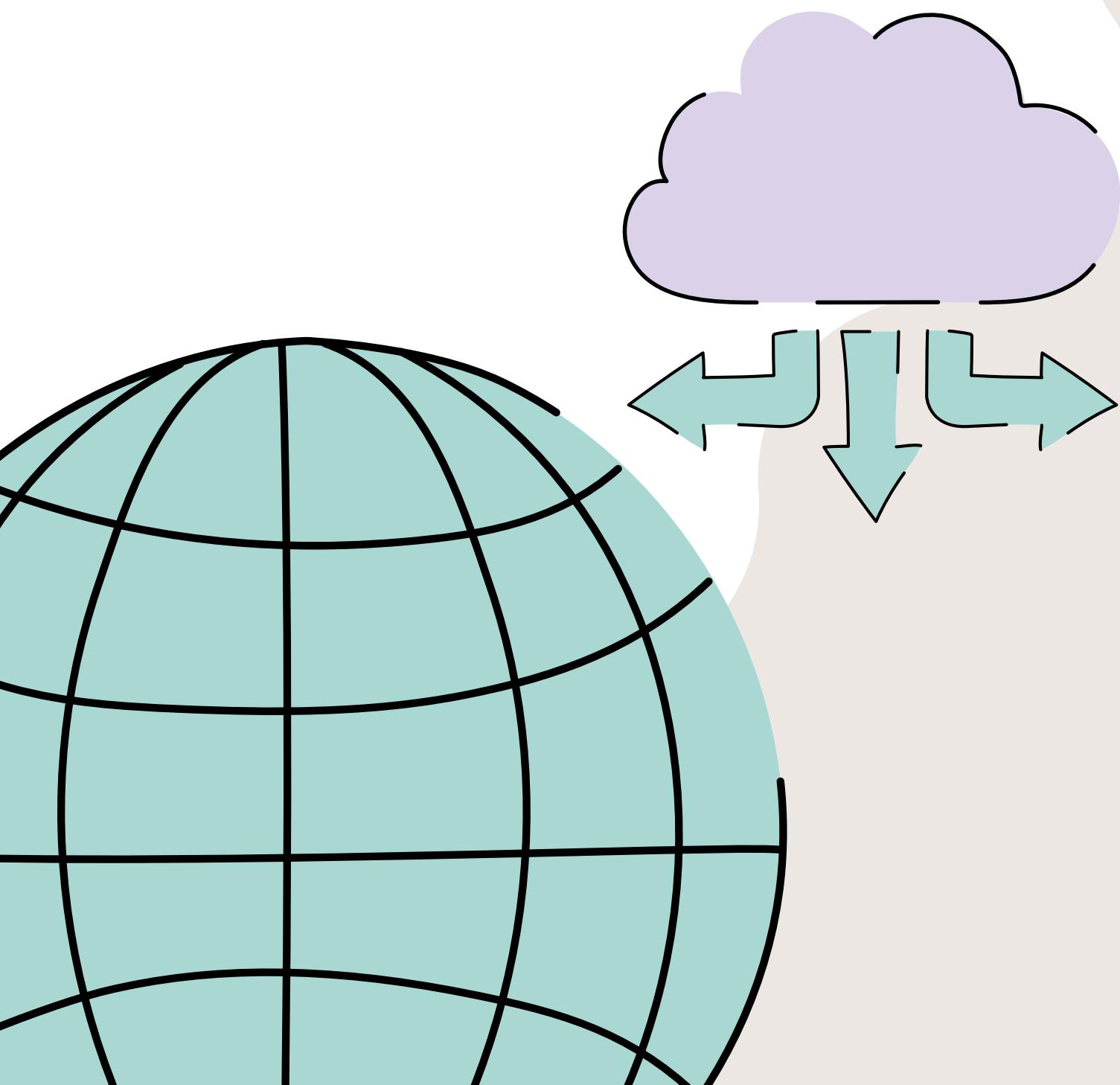
# Cons

- Tecnológicamente complejo
- Sin herramientas de monitoreo
- Dependencia en Zookeeper a no ser que se use el protocolo KRaft

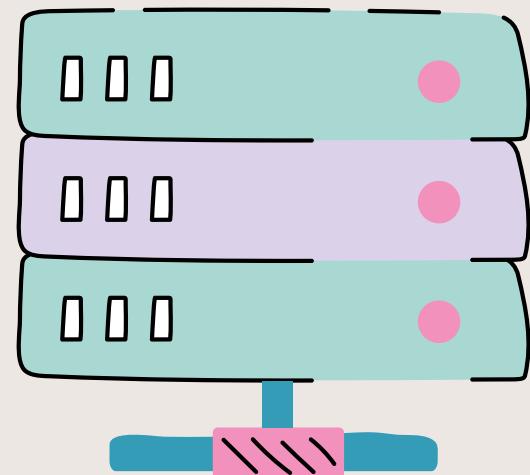


**versus**

# Casos de uso

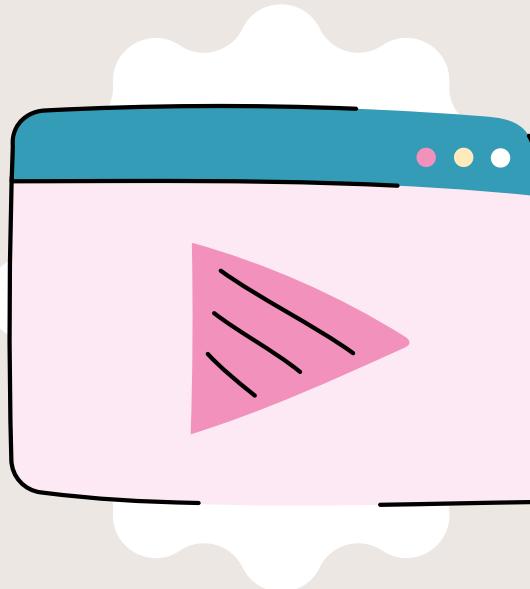


- Procesamiento de datos en tiempo real
- Mensajería
- Registro de eventos



# Casos de Aplicación en la Industria

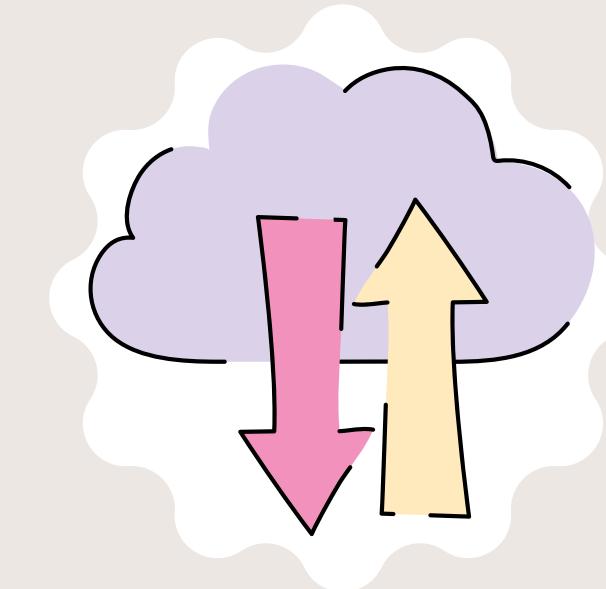
El 80% de las empresas en el Fortune 100 Companies usan Kafka



**Infraestructura**



**Bancos**



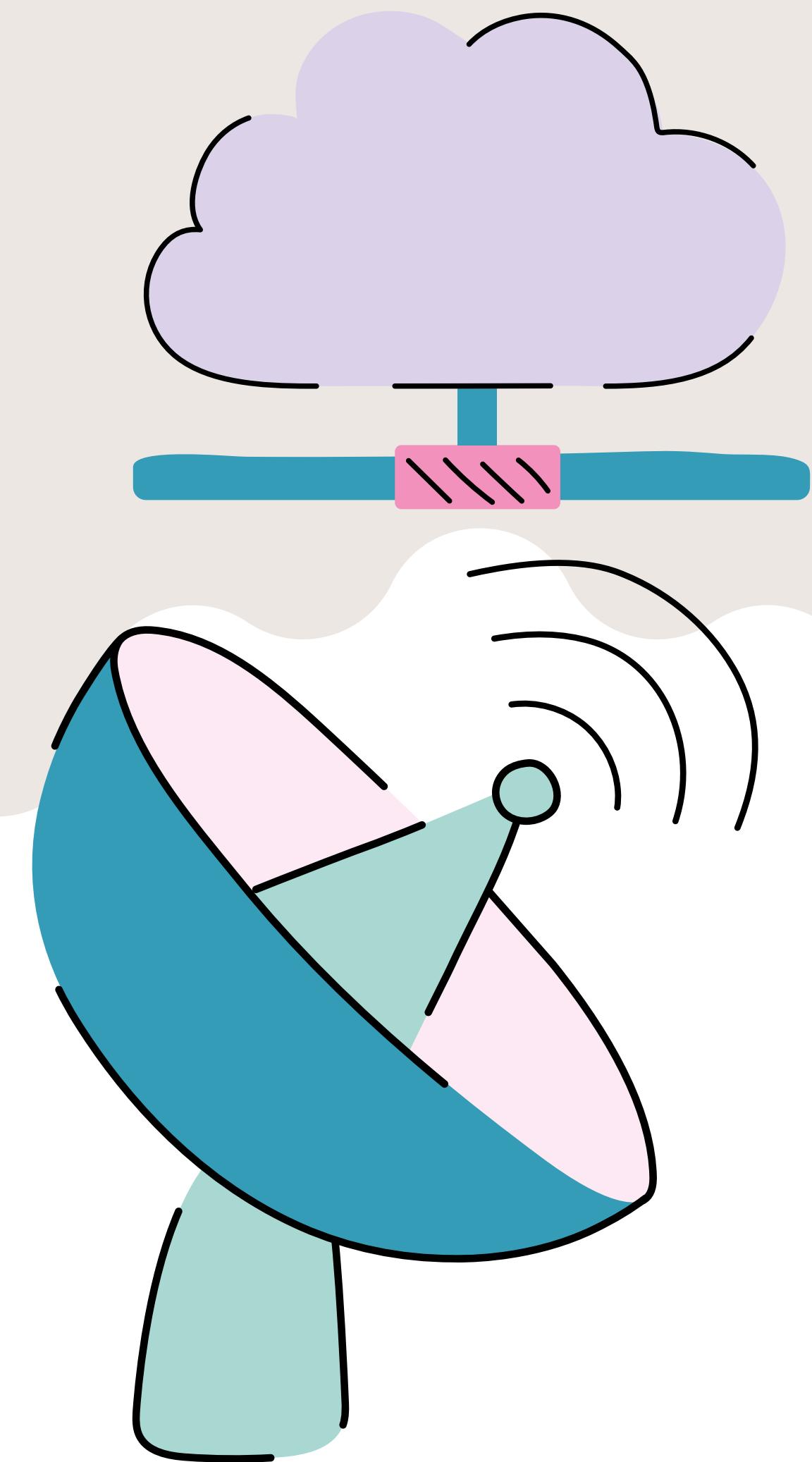
**Telecomunicación**



# Definicion

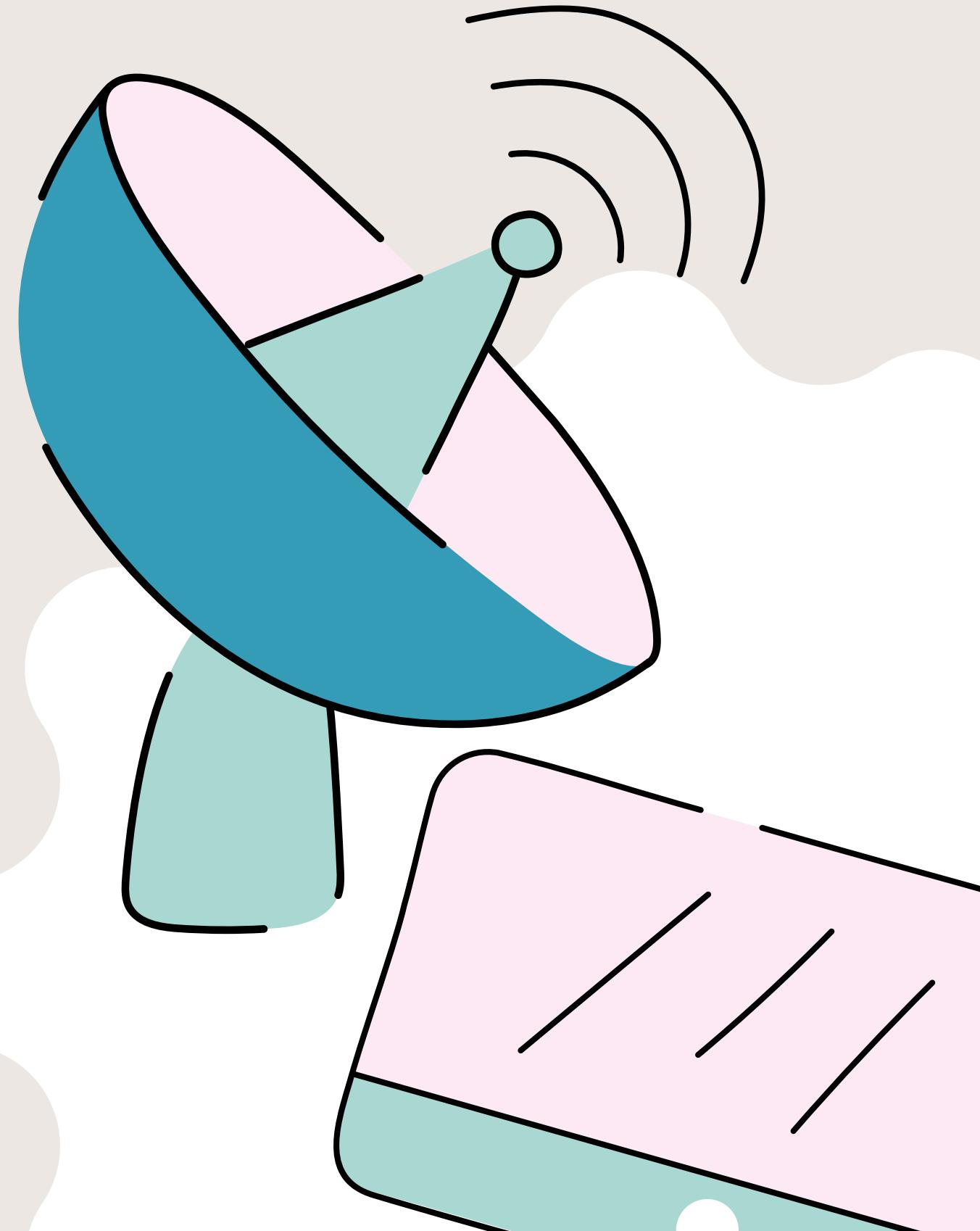
Apache Cassandra es un sistema de administración de base de datos noSql Open Source diseñado para soportar un alto volumen de información

Su nombre viene de la profeta Cassandra en mitología Troyana



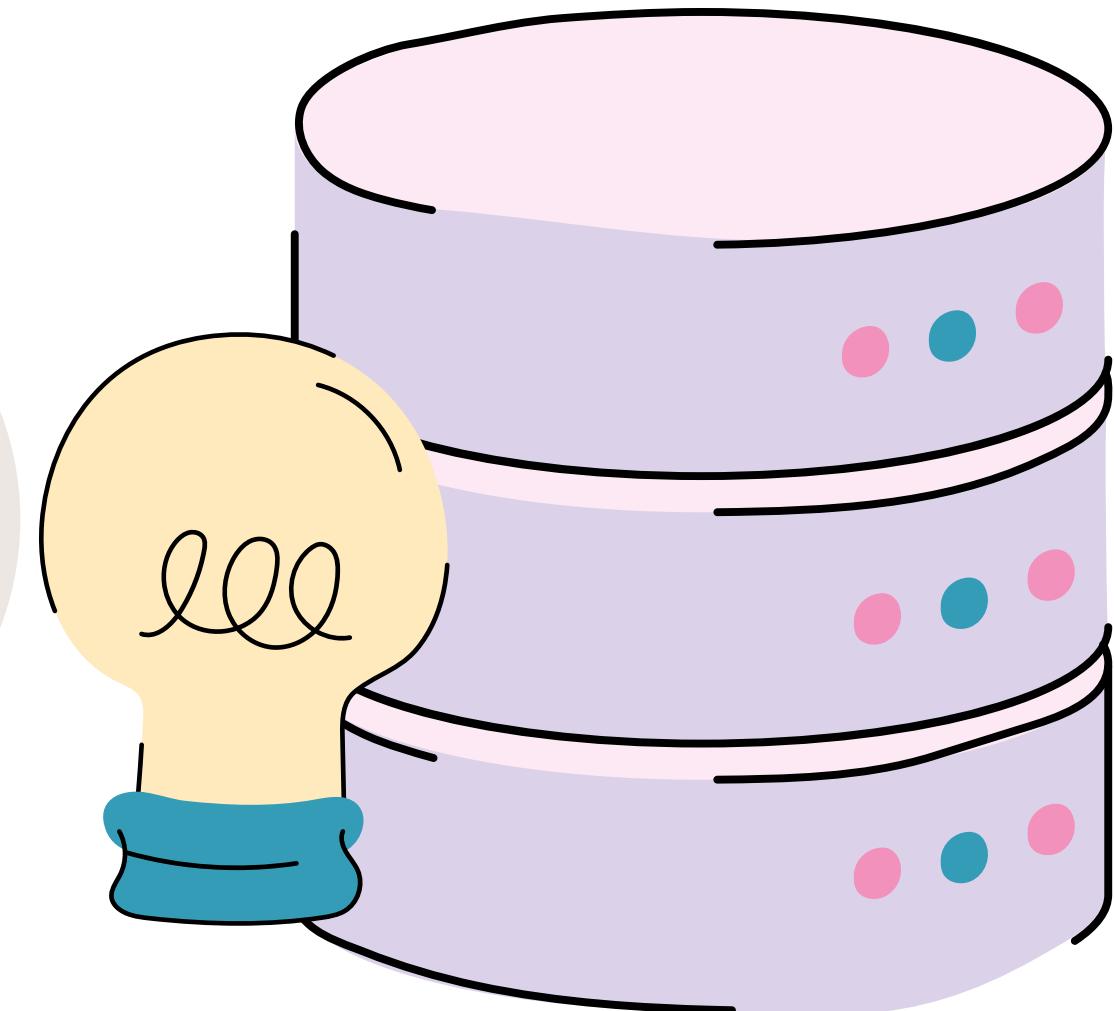
# Características

- NoSql basado en columnas
- Arquitectura distribuida con redundancia y capacidad de recuperación ante desastres
- Clasificación AP (disponibilidad y resistencia particiones)
- Compatible con Hadoop
- Alta escalabilidad
- Replicación de datos por nodos



# Historia y Evolución

- 2000: Avinash Lakshman y Prashant Malik desarrollan Cassandra en Facebook
- 2008: Cassandra se vuelve Open Source en Google Code
- 2009: Cassandra se une a Apache
- 2010: Cassandra se gradúa como un Proyecto de Alto Nivel en Apache
- 2011: Versión 1.0
- 2013: Versión 2.0
- 2015: Versión 3.0
- 2021: Versión 4.0
- 2025: Versión 5.0. Última versión estable



# Pros

- No tiene único punto de fallo
- Alta disponibilidad
- Gran capacidad de lectura
- Fácil escalabilidad

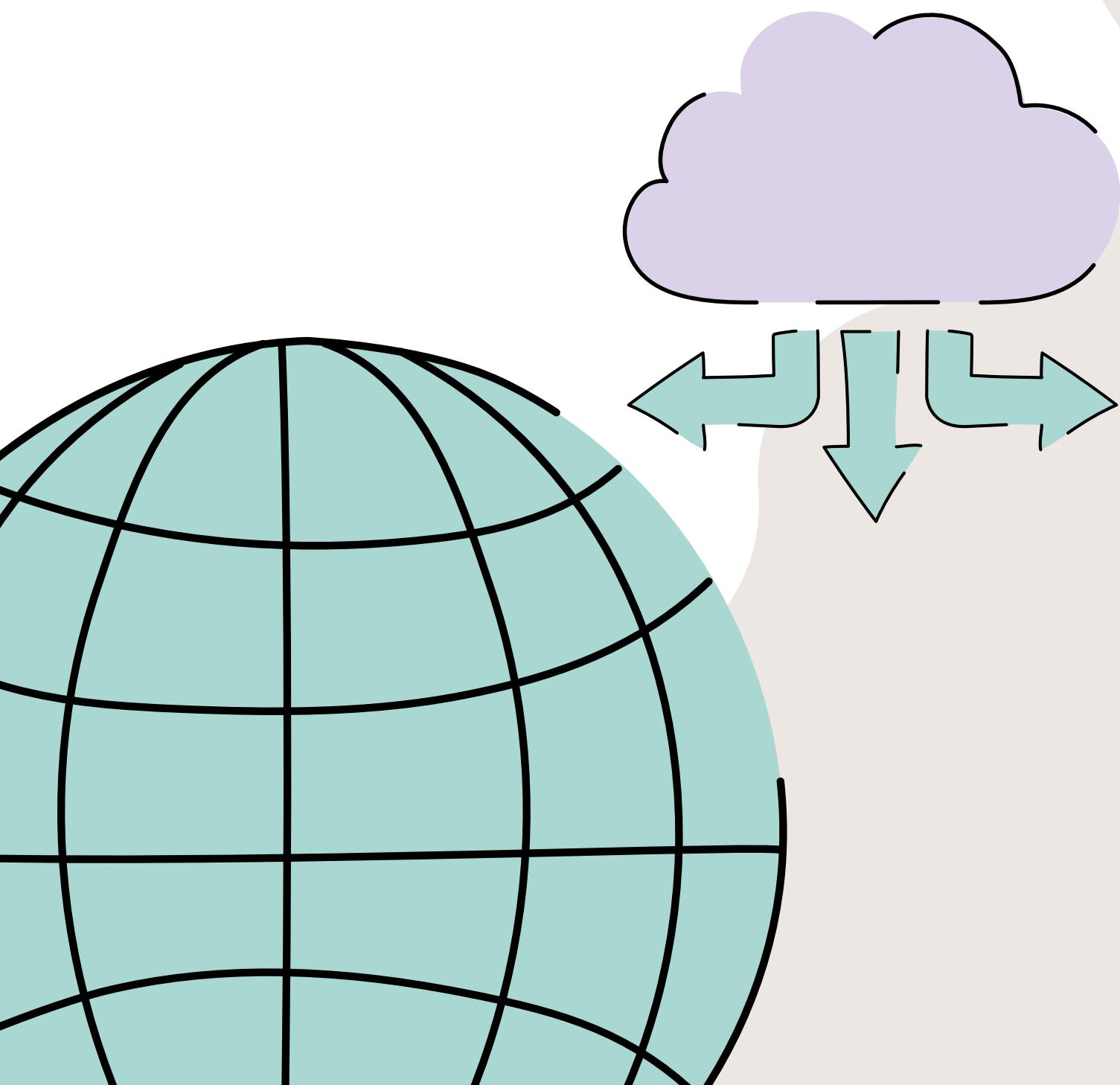
# Contras

- No soporta queries complejas
- No soporta Joins
- Consistencia eventual

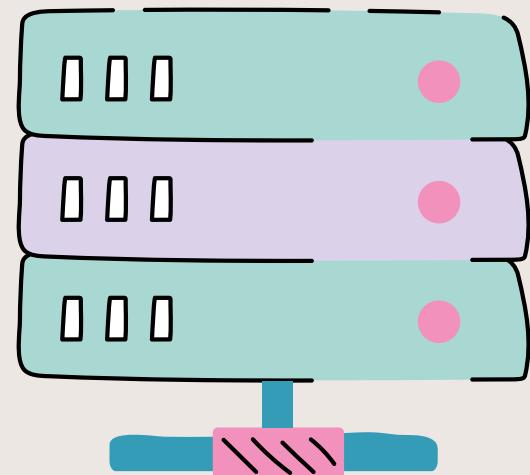


**versus**

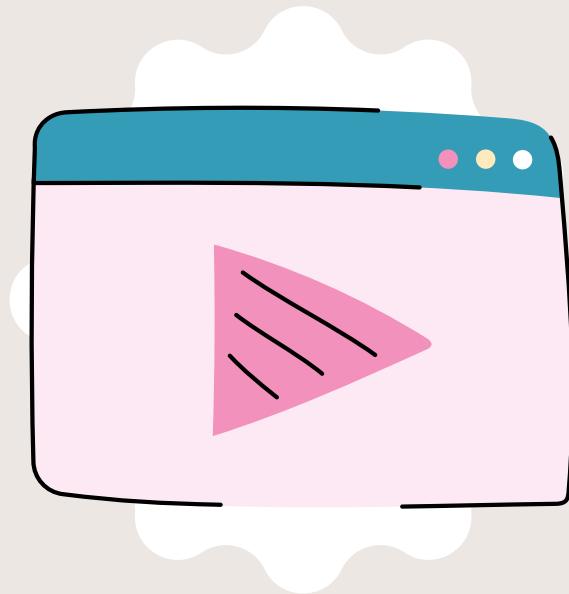
# Casos de uso



Cassandra es una opción viable en escenarios donde se requiera una base de datos NoSQL, especialmente para sistemas de alta lectura, como analítica web, plataformas de streaming o telemetría en tiempo real



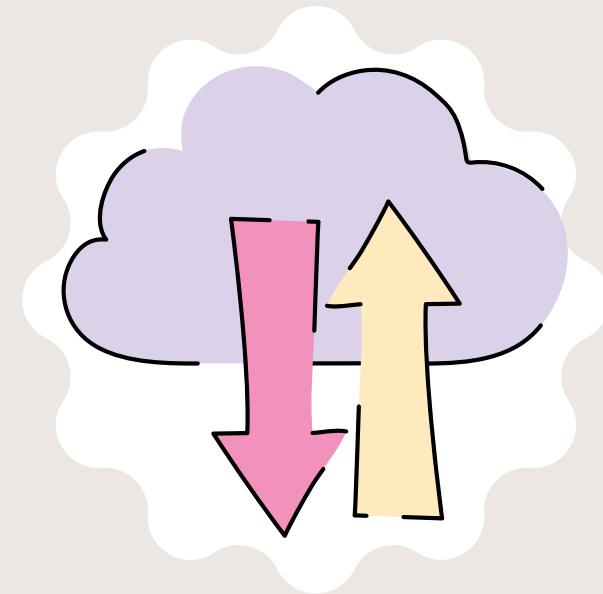
# Casos de Aplicación en la Industria



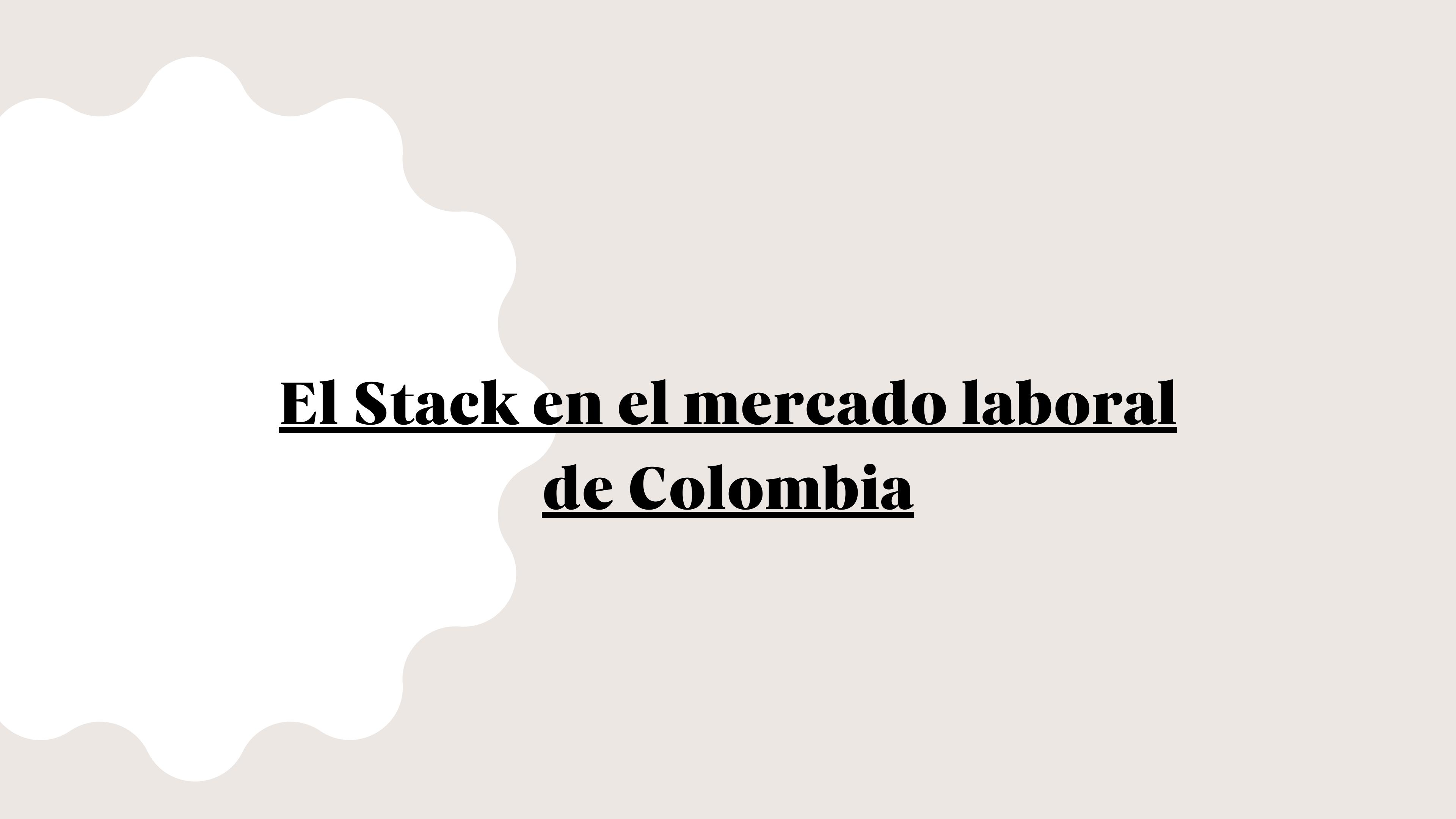
**Activision**



**Best Buy**

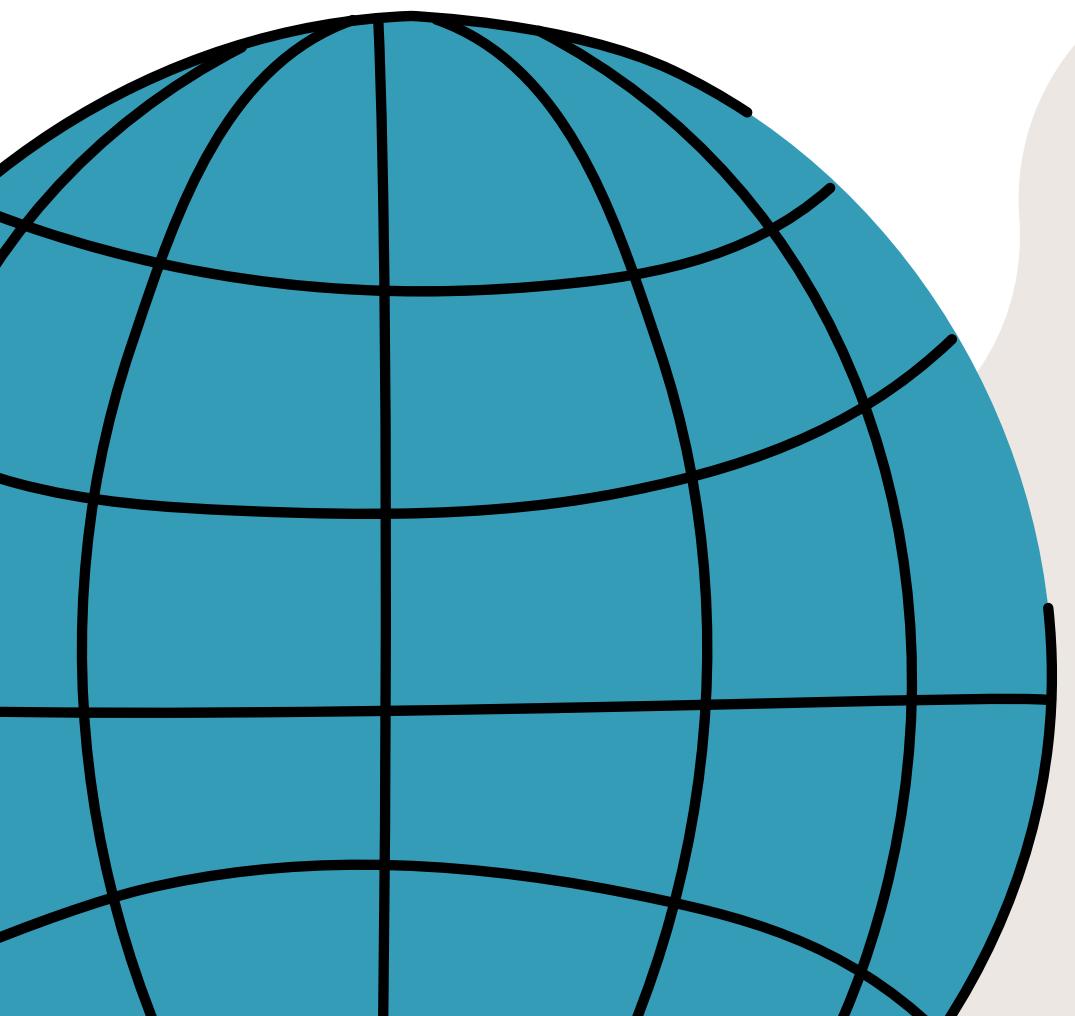


**Apple**



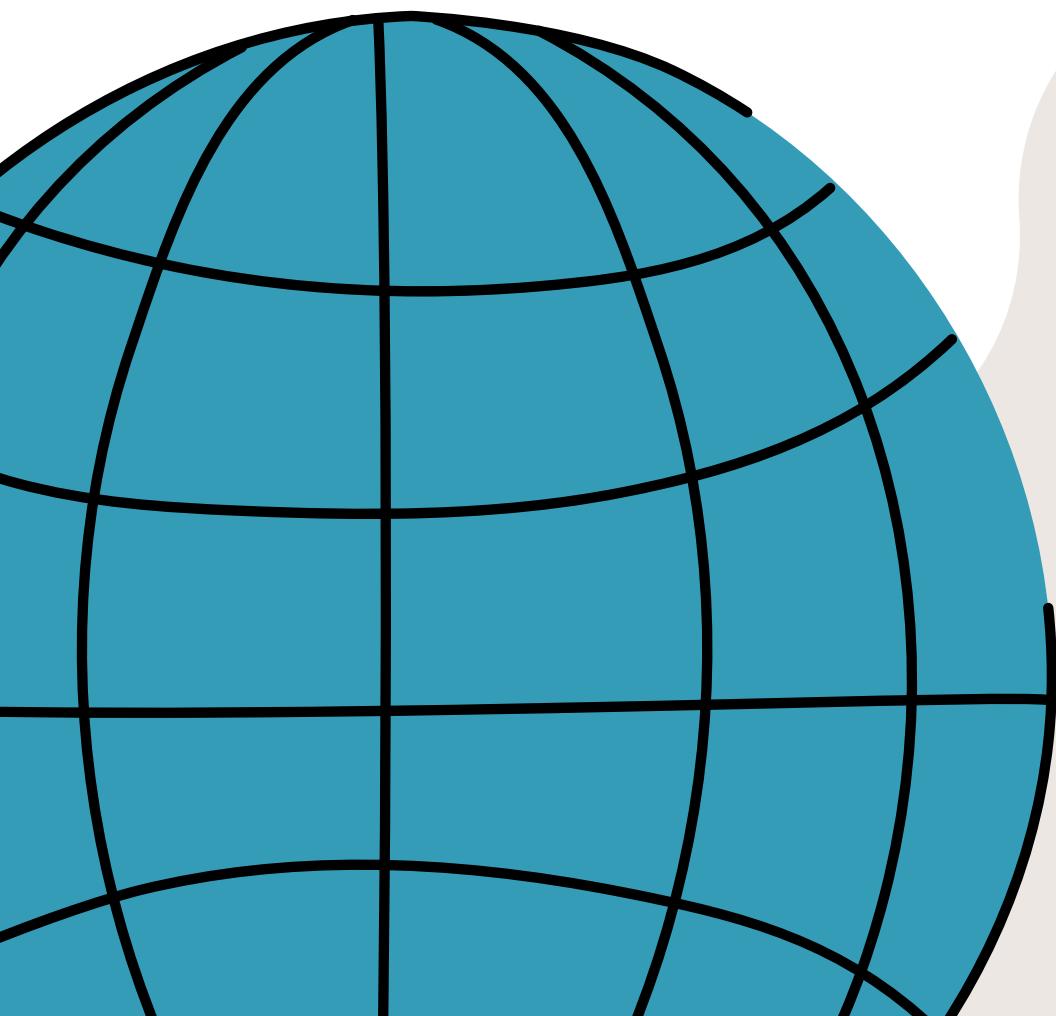
# **El Stack en el mercado laboral**

## **de Colombia**

- 
- EDA y Cola de eventos: Backend y Fullstack para móvil o web
  - Astro: Sin demanda para Astro en específico, demanda para Landing Pages y SaaS
  - Ruby y Rails: desarrollador Backend Junior, combinado con .NET, Node.Js o JavaScript
  - Kafka: Puestos de desarrollador Backend para Java
  - Cassandra: Nicho pequeño, principalmente para ingeniero de datos y desarrollador Backend

# Implementación del stack y diseño

# Descripción del sistema - Marketplace



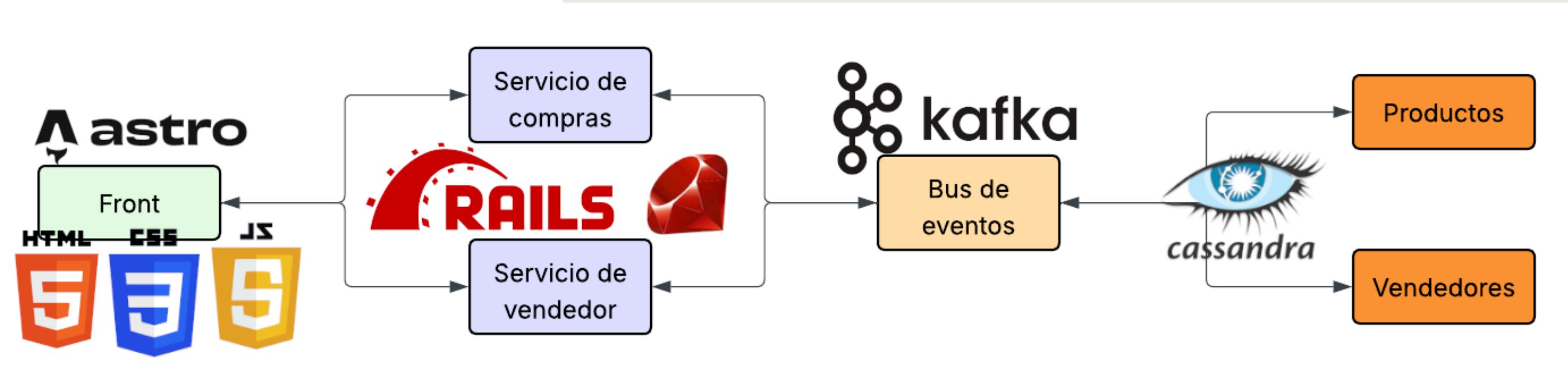
## Funcionalidades

- Compra de productos
- Administración de productos
- Notificaciones de ventas
- Resumen de ventas por producto
- Ganancia total

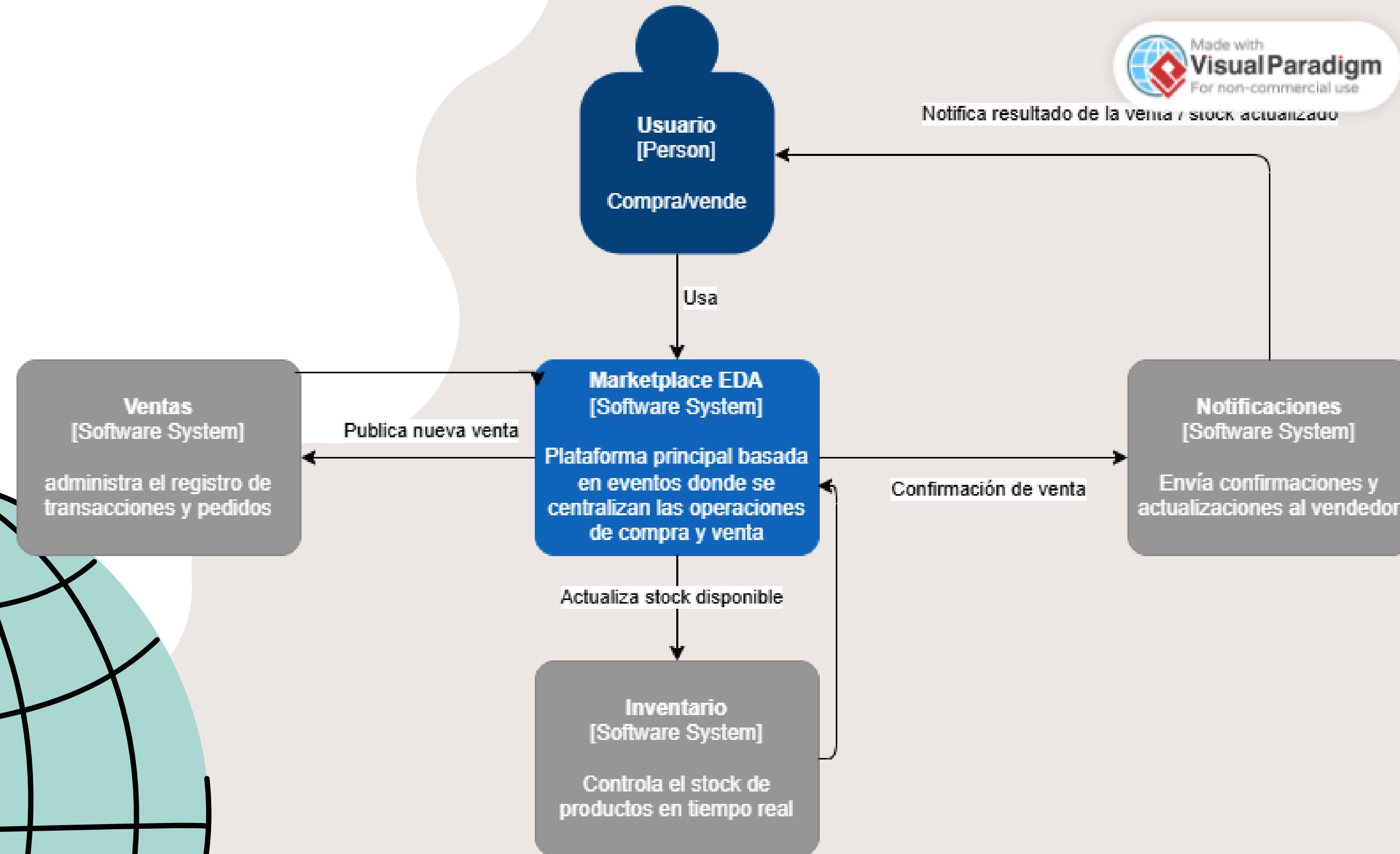
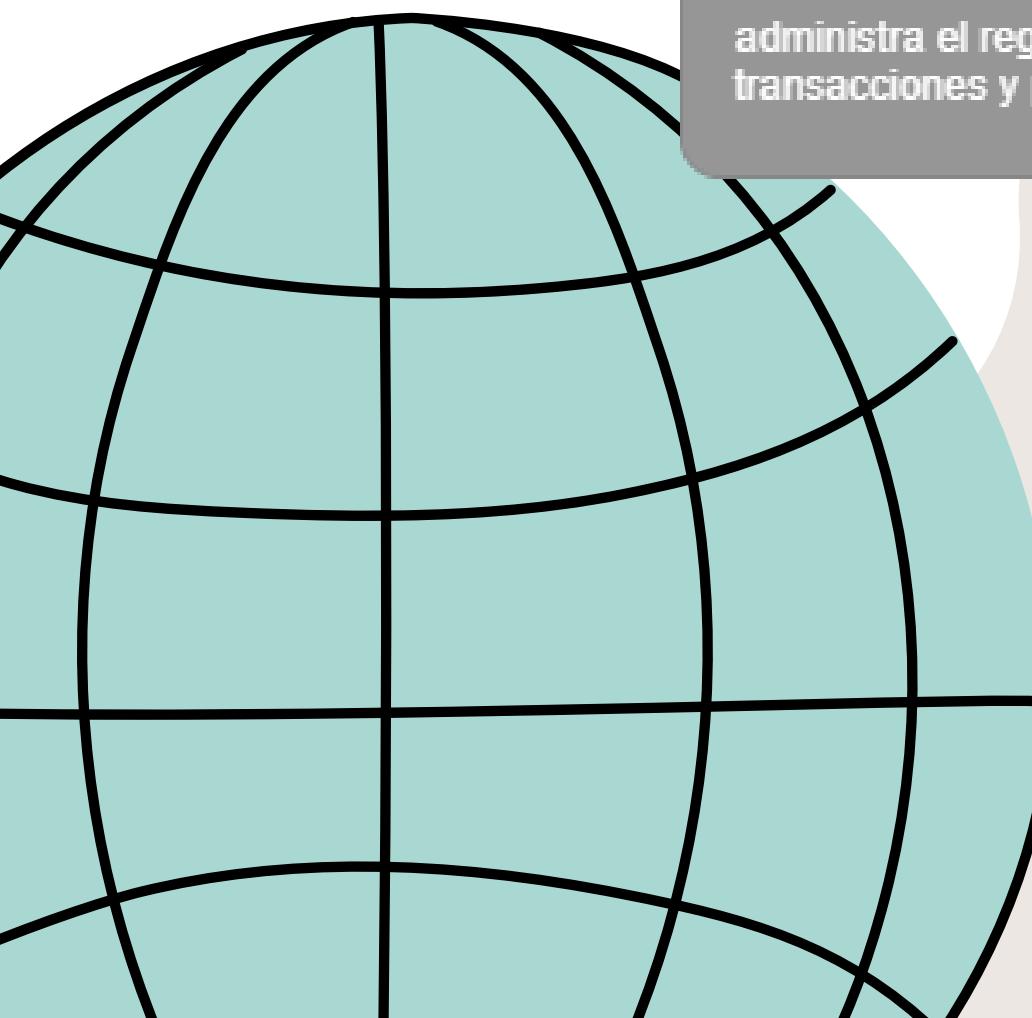
## Logica

- Frontend de Astro.js
- Ruby on Rails genera eventos
- Karafka consume los eventos generados
- Kafka genera queries a Cassandra

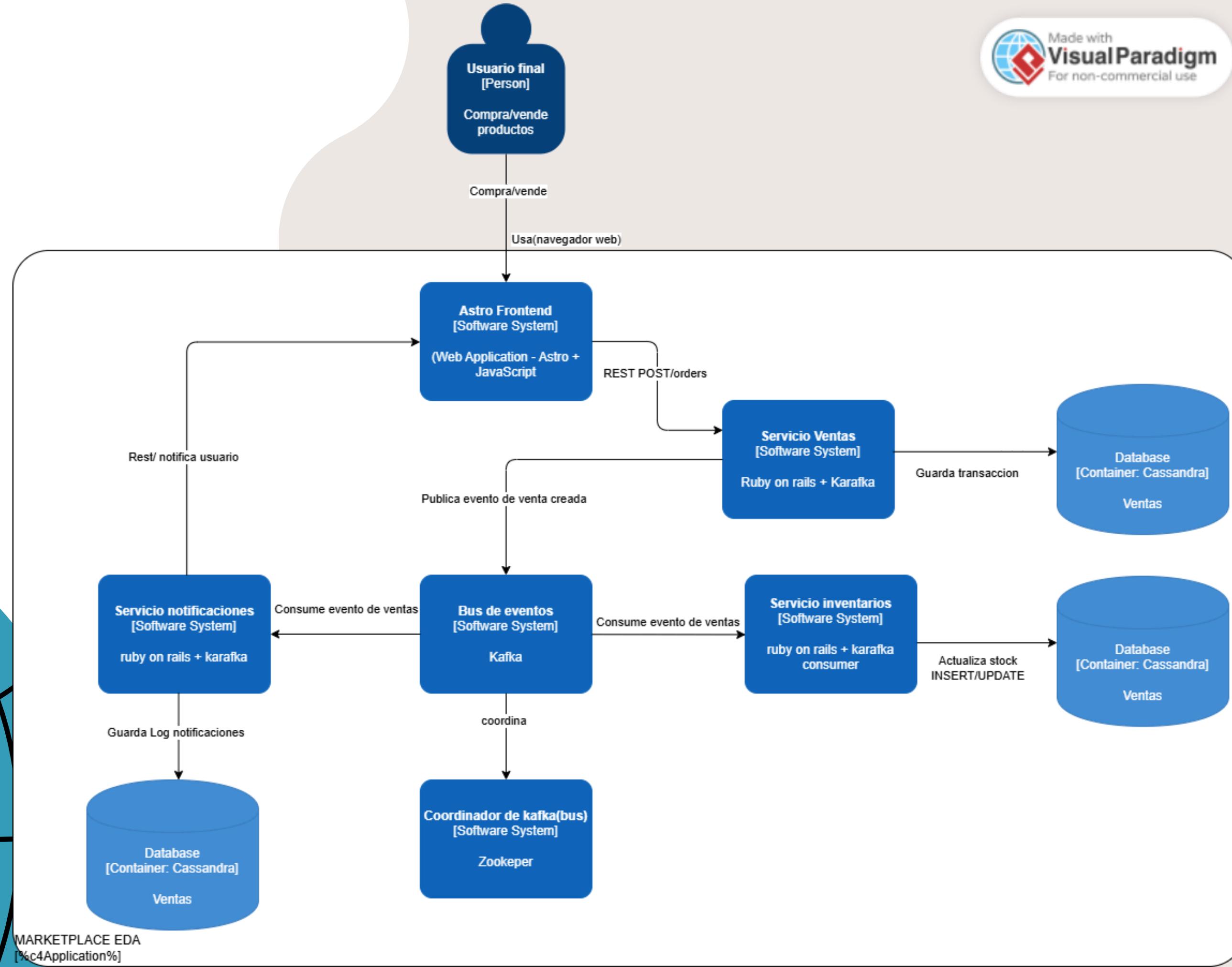
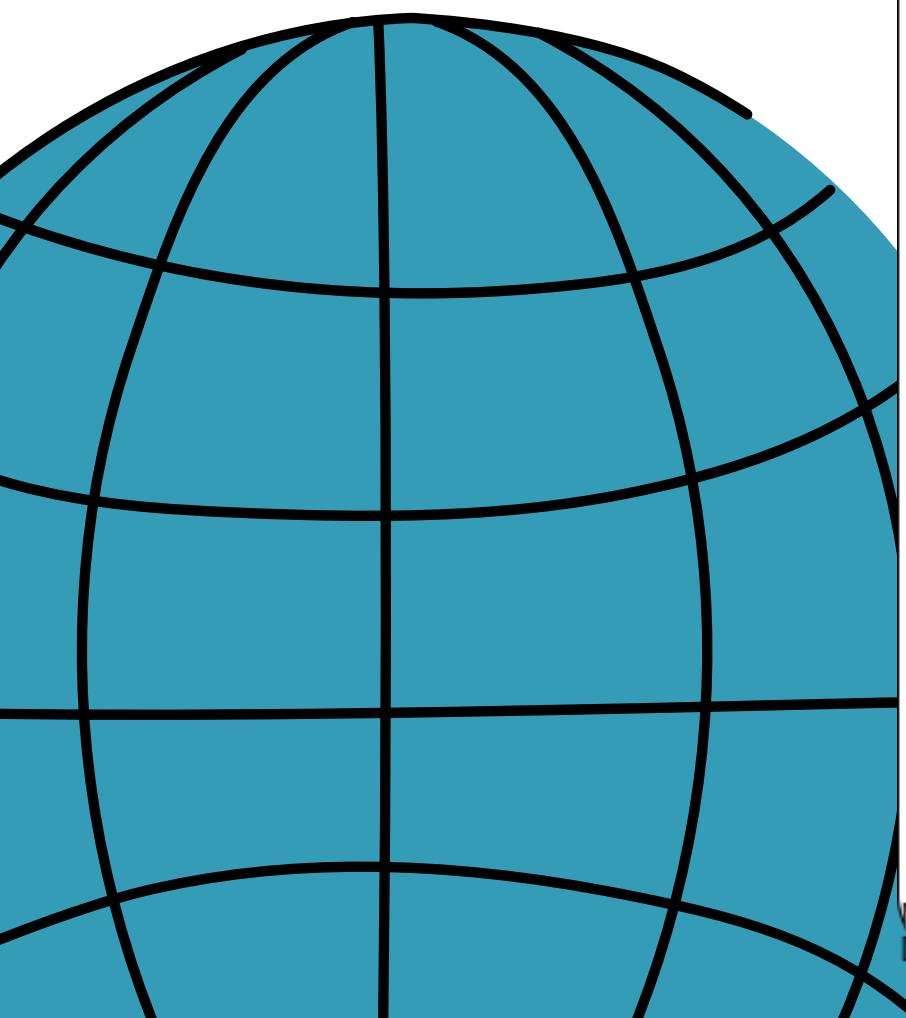
# Alto nivel



# C1



# C2



# C3



**Astro Frontend**  
[Component: Web app + astro + js]  
  
Aplicación web que envía solicitudes de compra o venta al backend.

Usa  
[HTTP POST/orders]

## Rails- Servicio de Ventas

**Ventas Controller**  
[Component: Ruby on Rails]  
  
Controlador REST, que recibe POST/orders del front y valida la información

**Ventas Service**  
[Component: Ruby]  
  
Contiene la logica de negocio de la venta, invoca al repositorio y al productor

Usa  
[%c4Technology%]

HTTP POST

**Kafka Producer**  
[Component: Kafka + kafka]  
  
Publica el evento venta\_creada en el bus de eventos (Kafka) para que otros servicios lo consuman

Uses  
[Send Ventas]

**Ventas Repository**  
[Component: Cassandra]  
  
Gestiona la persistencia de los datos de la venta en Cassandra.

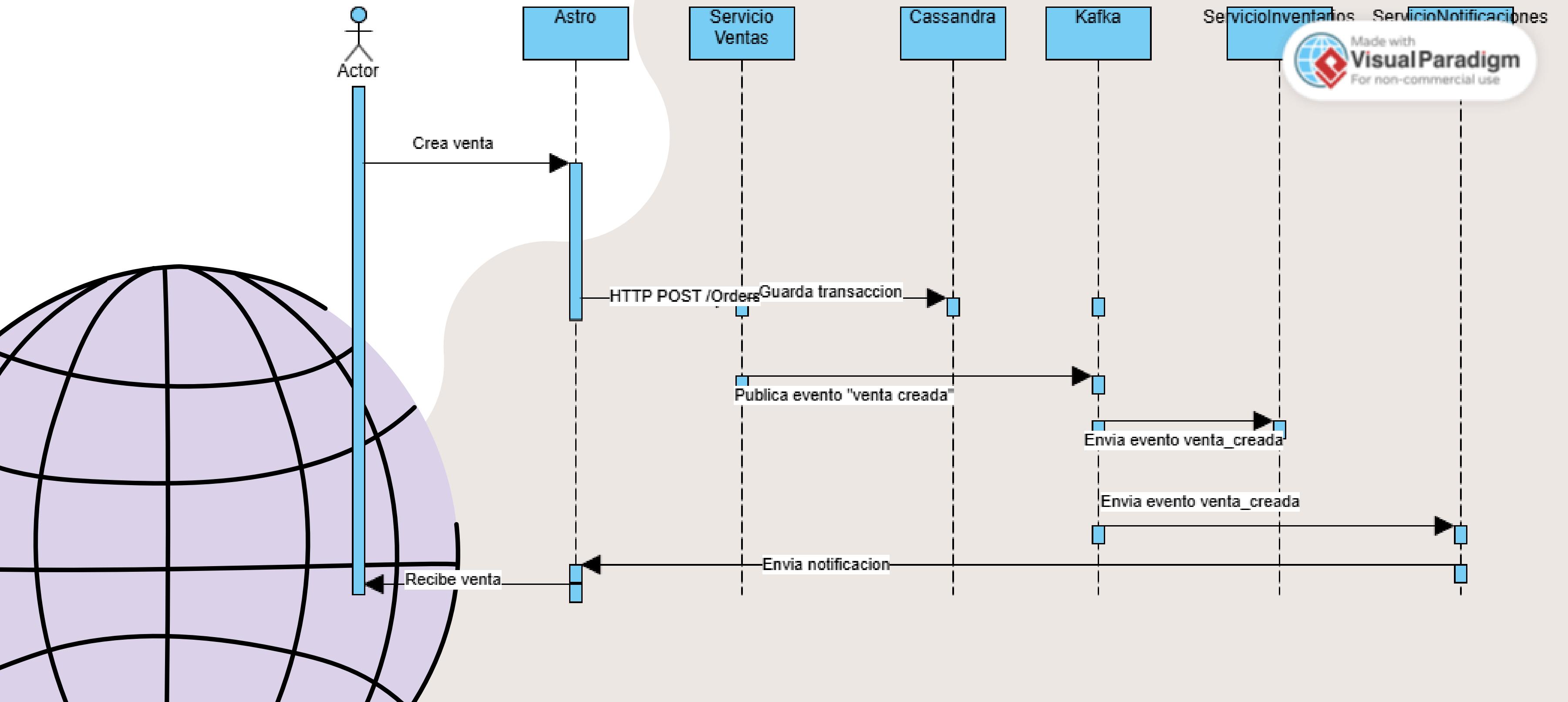
Usa  
[INSERT/UPDATE venta]

R/W Ventas  
[JSON]

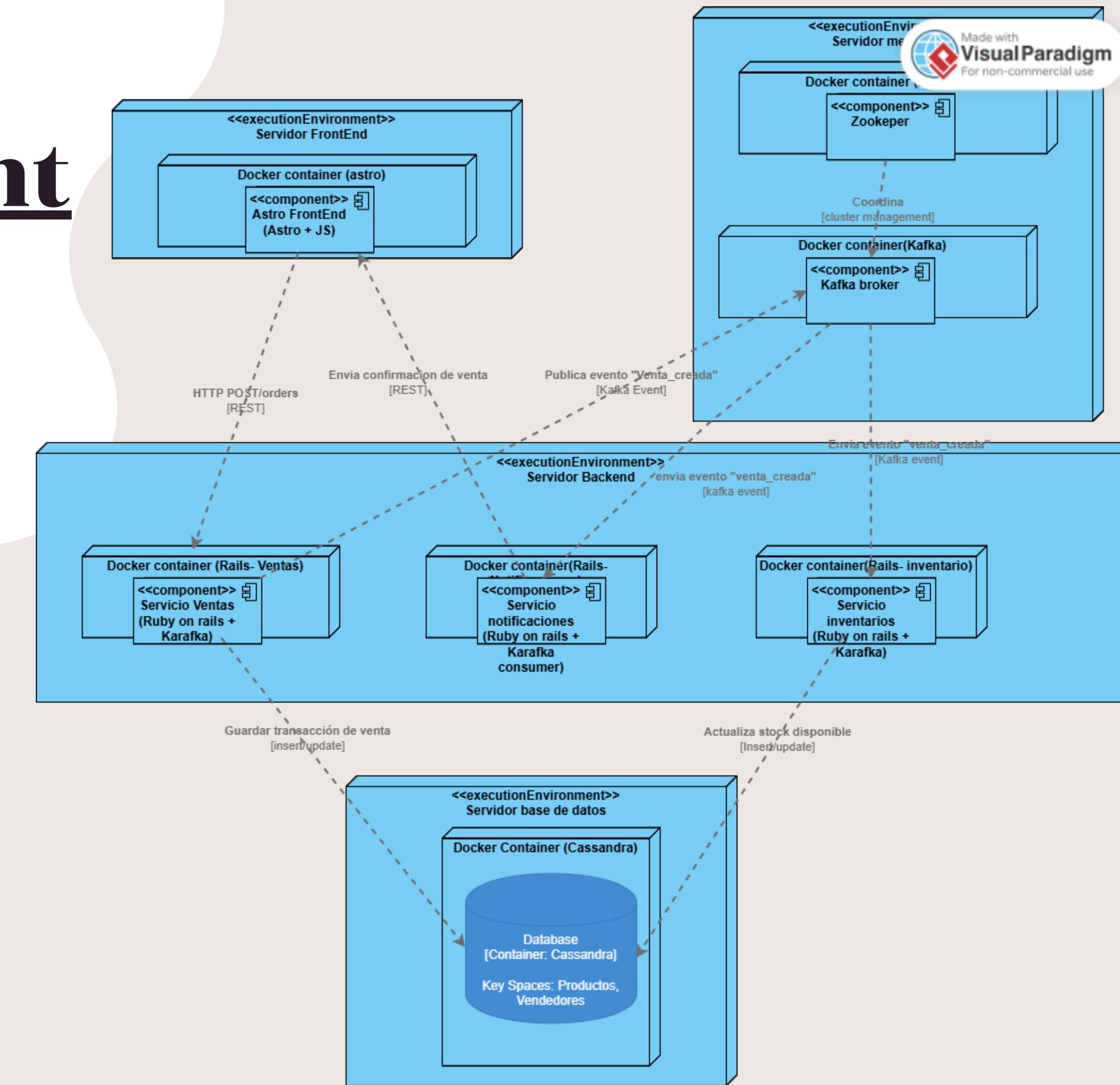
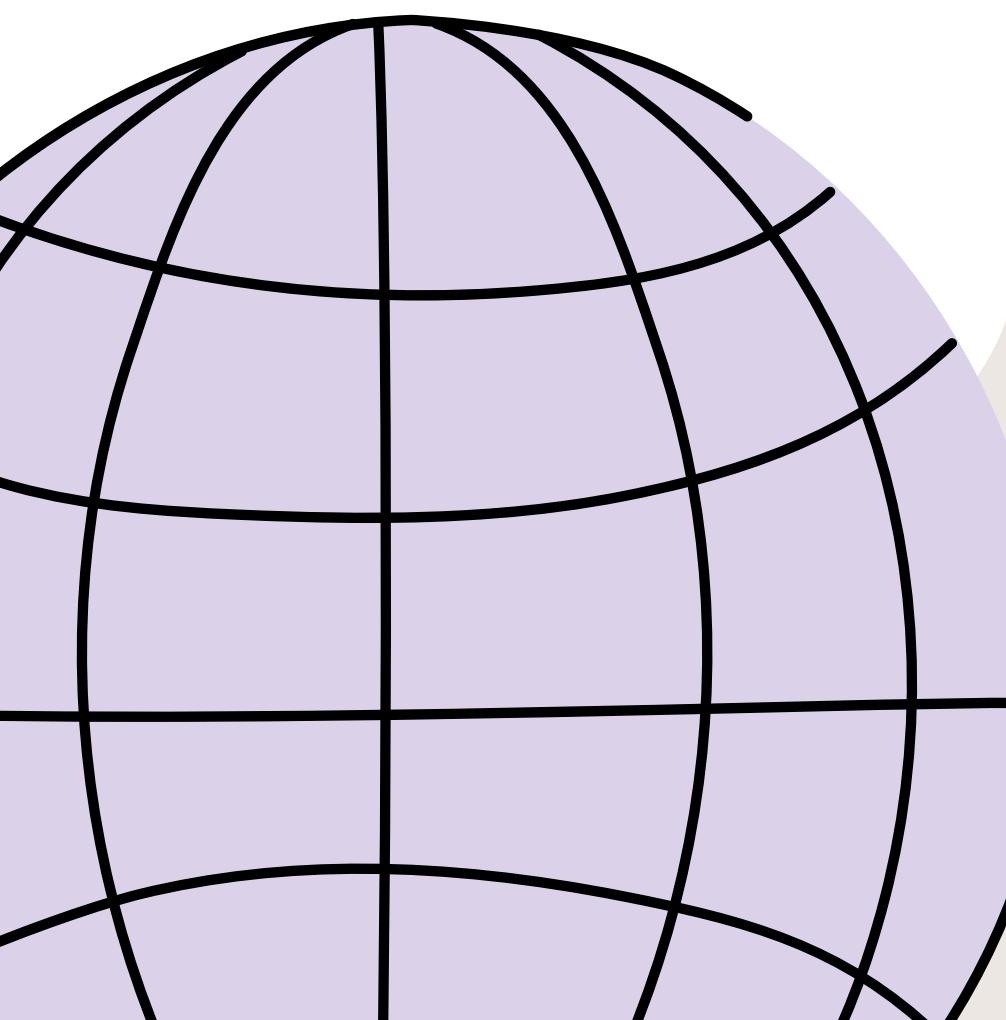
**Bus de eventos**  
[Component: Kafka]  
  
Canal de comunicación asíncrono donde se publican y consumen eventos.

**Database**  
[Container: Cassandra]  
  
Almacena las transacciones de ventas

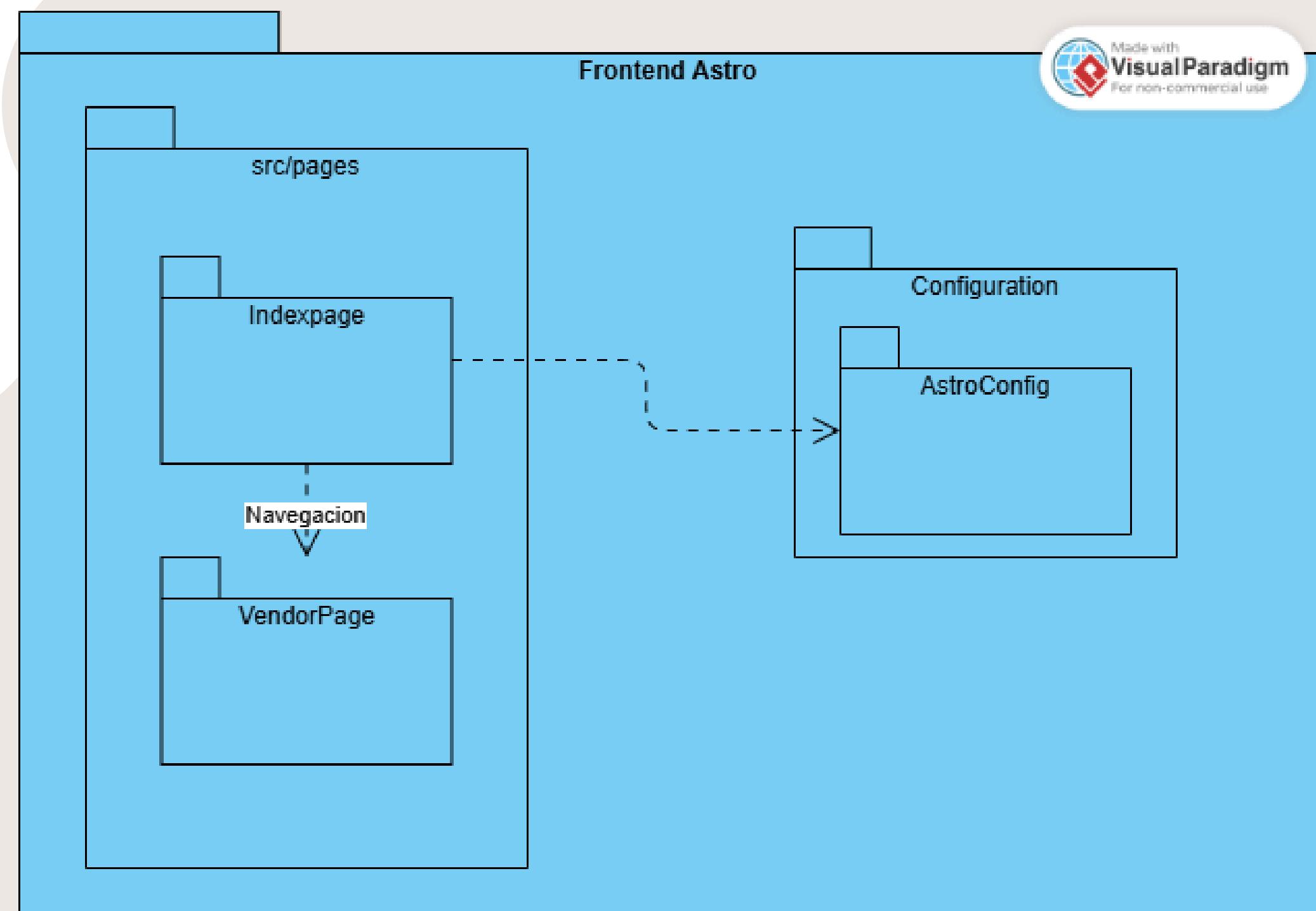
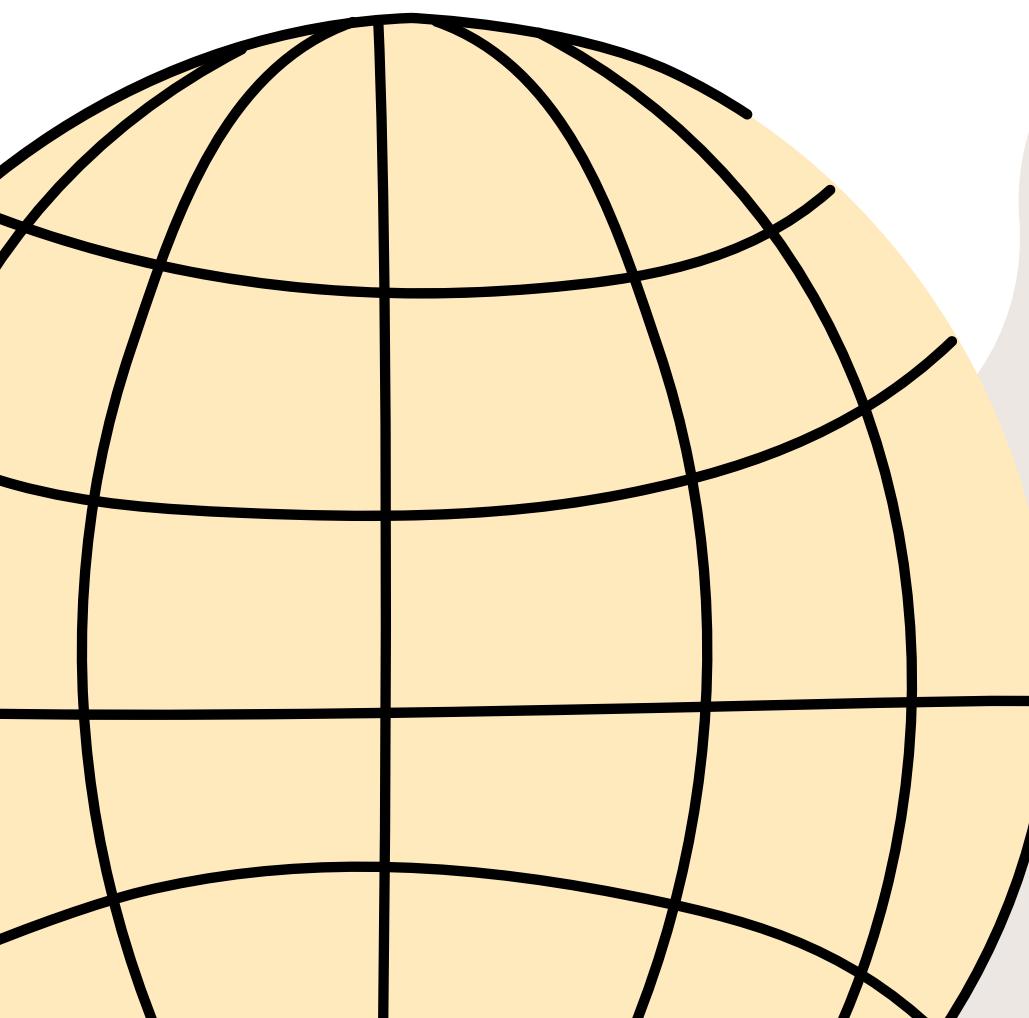
# C4 dynamic



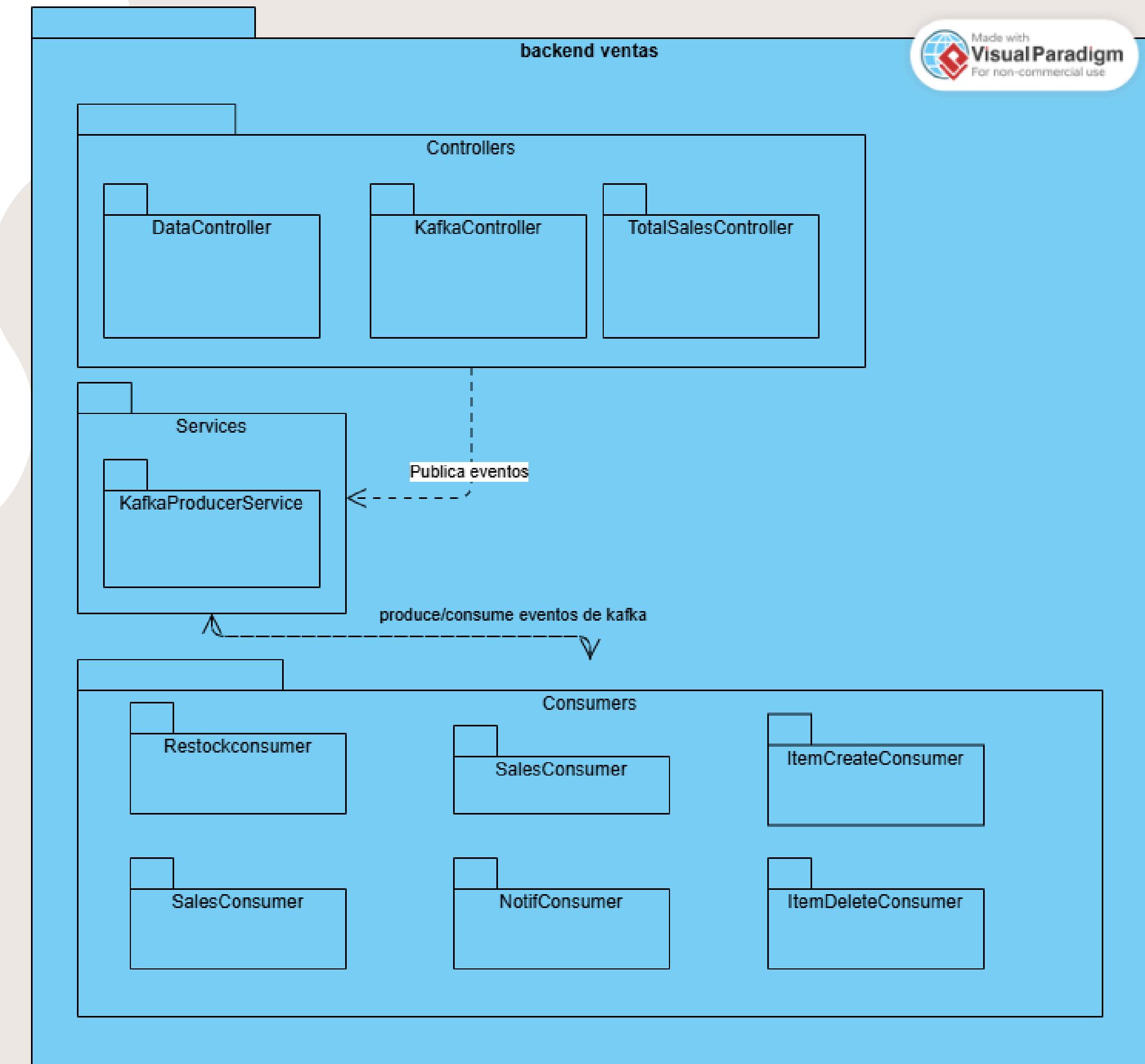
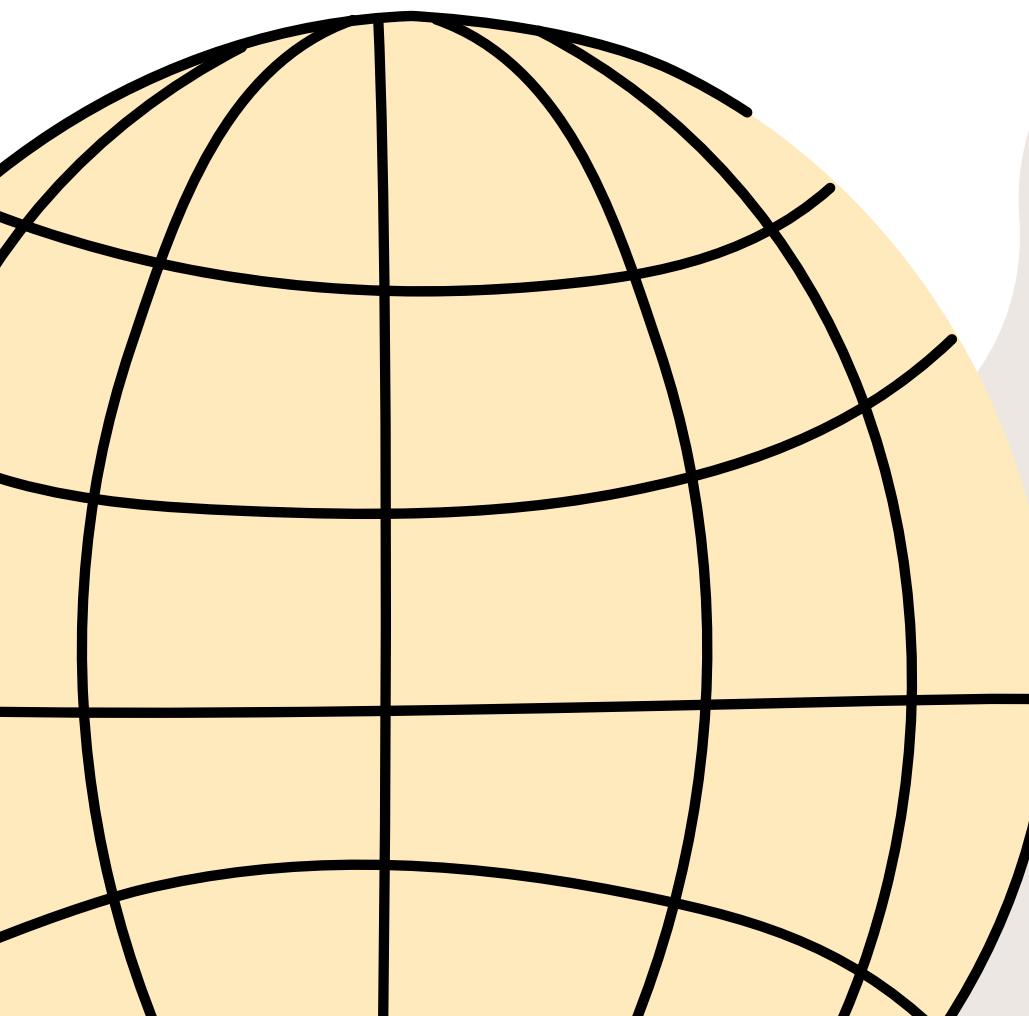
# C4 Deployment



# Diagrama De paquetes UML Front



# Diagrama De paquetes UML Back



# Referencias

- [https://en.wikipedia.org/wiki/Apache\\_Kafka](https://en.wikipedia.org/wiki/Apache_Kafka)
- <https://kafka.apache.org/>
- <https://www.altexsoft.com/blog/apache-kafka-pros-cons/>
- Hohpe, G., & Woolf, B. (2003). Enterprise Integration Patterns. Addison-Wesley.
- Richards, M. (2020). Software Architecture Patterns. O'Reilly Media.
- Fowler, M. (2017). Event-Driven Architecture. [martinfowler.com](http://martinfowler.com).
- Bass, L., Clements, P., & Kazman, R. (2022). Software Architecture in Practice (4th ed.). Addison-Wesley.
- Kleppmann, M. (2017). Designing Data-Intensive Applications. O'Reilly Media.
- Apache Software Foundation. (2024). Apache Kafka Documentation. <https://kafka.apache.org/documentation/>
- Confluent Inc. (2024). Event Streaming Platform Guide. <https://www.confluent.io/what-is-event-streaming/>
- Netflix TechBlog. (2023). How Netflix Uses Kafka for Real-Time Event Processing. <https://netflixtechblog.com/>
- Uber Engineering. (2022). Building Reliable Event-Driven Systems at Scale. <https://eng.uber.com/>
- Namiot, D., & Sneps-Sneppe, M. (2014). On Microservices Architecture. International Journal of Open Information Technologies, 2(9), 24–27.
- Vaughn, V. (2016). Implementing Domain-Driven Design. Addison-Wesley. (Capítulos sobre Event Sourcing y CQRS).
- Junco, R. (2024, octubre 9). Work Queues: The Simplest Form of Batch Processing. System Design Classroom. <https://newsletter.systemdesignclassroom.com/p/work-queues-the-simplest-form-of>
- Iberasync. (s. f.). Buses de mensajes y colas de mensajes en sistemas distribuidos cloud. Iberasync. Recuperado octubre 2025, de <https://iberasync.es/buses-de-mensajes-y-colas-de-mensajes-en-sistemas-distribuidos-cloud/>
- Redpanda Data. (2024, junio 18). Enterprise Messaging vs. Event Streaming. Redpanda Blog. <https://www.redpanda.com/blog/enterprise-messaging-vs-event-streaming>

# Referencias

- [https://en.wikipedia.org/wiki/Ruby\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Ruby_(programming_language))
- <https://www.geeksforgeeks.org/ruby/ruby-programming-language/>
- <https://pangea.ai/resources/best-practices-ruby>
- [https://en.wikipedia.org/wiki/Ruby\\_on\\_Rails](https://en.wikipedia.org/wiki/Ruby_on_Rails)
- <https://www.codica.com/blog/pros-and-cons-of-ruby-on-rails-for-web-development/>
- [https://en.wikipedia.org/wiki/Apache\\_Cassandra](https://en.wikipedia.org/wiki/Apache_Cassandra)
- <https://stackoverflow.com/questions/2634955/when-not-to-use-cassandra>
- <https://medium.com/@vinciabhinav7/cassandra-part-2-use-cases-alternatives-and-drawbacks-a152dce60c6b>
- [https://es.wikipedia.org/wiki/Astro\\_\(framework\)](https://es.wikipedia.org/wiki/Astro_(framework))
- <https://astro.build/>
- <https://www.linkedin.com/>

# Demo del prototipo

