

# Taller CQRS Cassandra, RubyOnRails, GraphQL y Android

Presentación del taller #2

Por Santiago Yáñez, Hermann  
Hernandez, Alejandro Suárez.  
Lunes, 30 de septiembre de 2024.



# Contenidos

---



**01** Introducción

---

**02** Patrón CQRS

---

**03** Android

---

**04** GraphQL

---

**05** RubyOnRails

---

**06** Cassandra

---

**07** Matrices

---

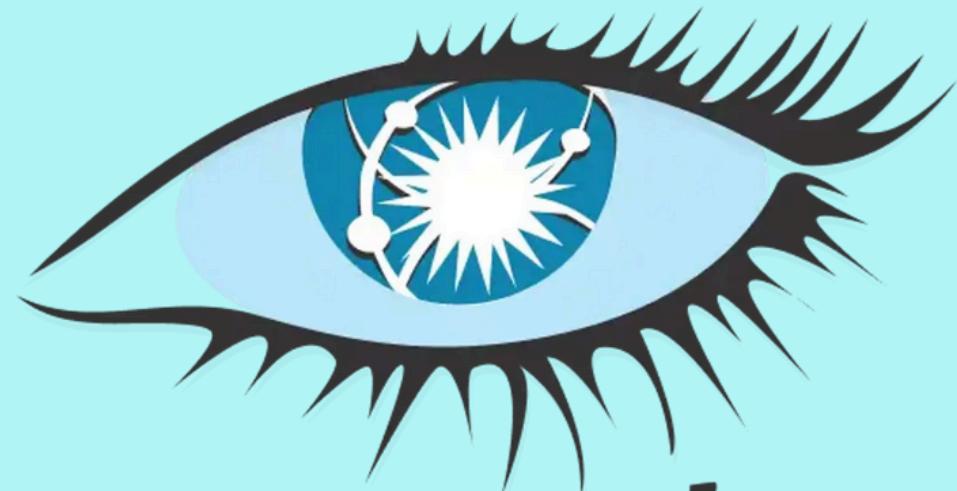
**08** Diagramas

---

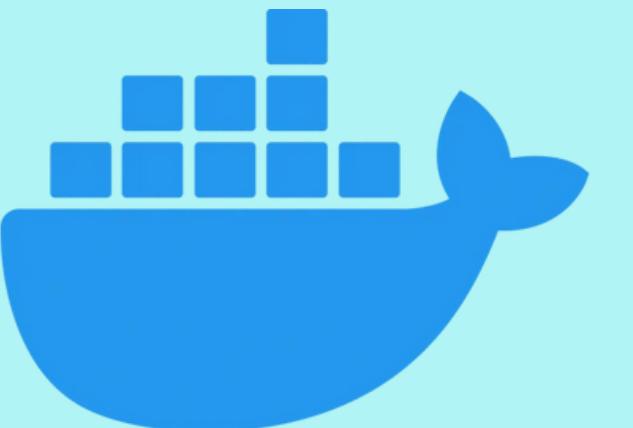
**09** Aprendizajes

---

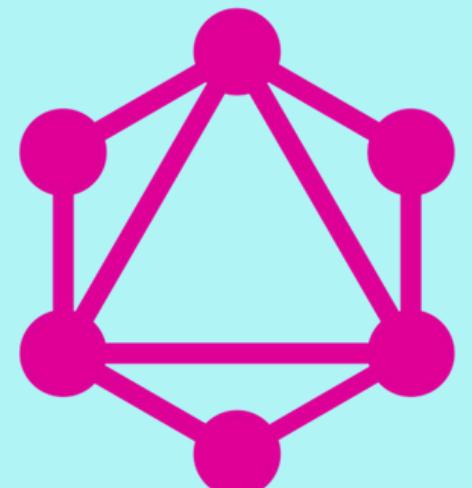
# Introducción



*cassandra*



docker®



GraphQL



# Definición

Separa las operaciones de lectura (consultas) y escritura (comandos) en sistemas complejos. Esto permite:

**Manejar modelos de datos y lógica de negocio diferentes para cada operación.**

**Optimizar independientemente la lectura y escritura.**

**Mejorar el rendimiento y la escalabilidad en sistemas distribuidos.**

# CQRS

Command Query Responsibility Segregation

# Características

# CQRS

Command Query Responsibility Segregation

1

## Separación de responsabilidades

Las consultas no modifican el estado, solo devuelven datos, mientras que los comandos cambian el estado.

2

## Modelos de datos separados

Cada operación usa modelos optimizados para lectura o escritura, mejorando el rendimiento.

3

## Event Sourcing

A menudo combinado con CQRS para registrar detalladamente los cambios en el estado, mejorando la trazabilidad.

4

## Escalabilidad

Permite escalar consultas y comandos de forma independiente según sus necesidades tecnológicas.

# Historia



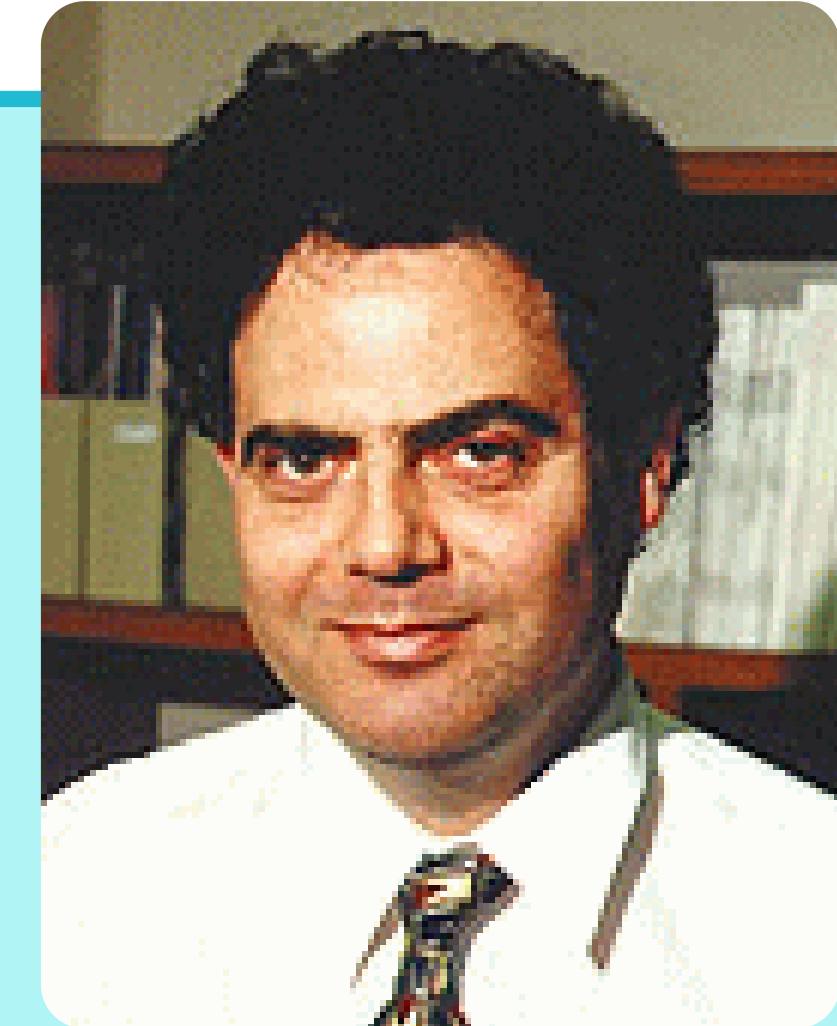
Basado en Command Query Separation (CQS) de Bertrand Meyer, que separa métodos en comandos (modifican estado) y consultas (devuelven datos).



Popularizado por Martin Fowler y Greg Young en la década de 2010



La evolución del patrón responde a las complejidades del mundo digital, donde lectura y escritura requieren optimizaciones distintas.



Crecimiento en entornos con grandes volúmenes de datos y necesidad de rendimiento eficiente.

# Ventajas

## Escalabilidad

Permite escalar de manera independiente las operaciones de lectura y escritura, lo que optimiza el uso de recursos según las necesidades del sistema.

## Separación de responsabilidades

La división entre comandos y consultas simplifica el desarrollo y mantenimiento, ya que cada uno puede evolucionar de forma independiente.

## Optimización específica

Facilita el uso de diferentes tecnologías o modelos de datos para optimizar el rendimiento de lectura y escritura.

# Desventajas

## Dificultad de sincronización

La sincronización entre los modelos de lectura y escritura puede ser complicada, especialmente si se utilizan tecnologías diferentes.

## Coste de implementación

- Requiere más esfuerzo y recursos para su implementación en comparación con arquitecturas más tradicionales como CRUD.

## No siempre es necesario

En sistemas simples o de baja carga, la adopción de CQRS puede ser excesiva, añadiendo complejidad sin proporcionar beneficios claros.

# CQRS

Command Query Responsibility Segregation



Ideal para plataformas con más consultas que comandos, como e-commerce (búsquedas frecuentes, pocas transacciones).

# CQRS

Command Query Responsibility Segregation

## Casos de uso

Adecuado para alta disponibilidad y rápida respuesta en consultas, como sistemas de seguimiento en tiempo real.

Facilita un control estricto del acceso a datos, separando y optimizando lectura y escritura.

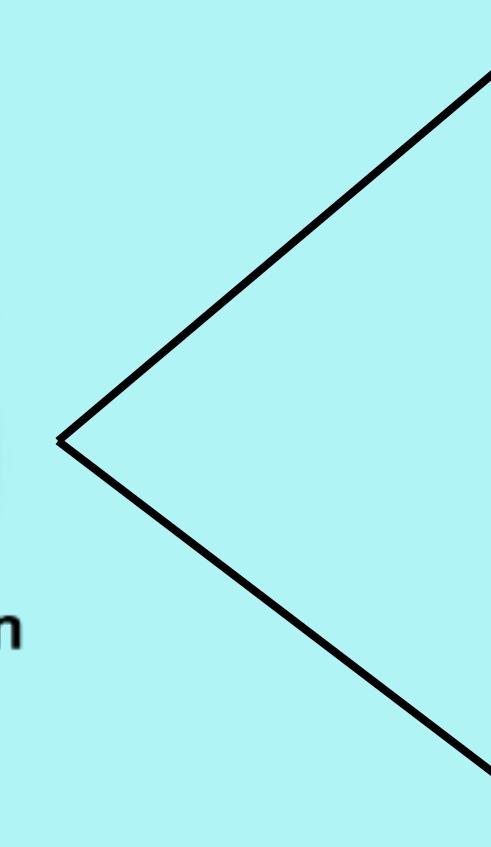
## Casos de aplicación

**CQRS**

Command Query Responsibility Segregation

ebay

amazon



# Definición

Android es un sistema operativo basado en Linux, diseñado para teléfonos móviles y otros dispositivos, que permite el desarrollo sencillo de aplicaciones. Es completamente libre y gratuito, lo que facilita su adopción en el mercado.

**Permite el desarrollo con Dalvik (variante de Java).**

**Libre y gratuito, sin licencias.**

**Más de 2,500 millones de dispositivos activos.**



# Características



- 1** Accesible para desarrolladores, diseñadores y fabricantes.
- 2** Protección desde el primer uso con Google Play Protect y actualizaciones de seguridad regulares.
- 3** Los usuarios tienen control sobre sus datos y opciones de privacidad accesibles.
- 4** Incluye un navegador web basado en WebKit.

# Historia



Android fue fundada en octubre de 2003 por Andy Rubin, Rich Miner, Nick Sears y Chris White, inicialmente como un sistema operativo para cámaras digitales antes de enfocarse en teléfonos inteligentes.



Android recibe su nombre del apodo de su creador, Andy Rubin, por su afición a los robots.



Existe oficialmente desde 2008, desarrollado por Google; fue adquirido por Google en 2005 por 50 millones de dólares.

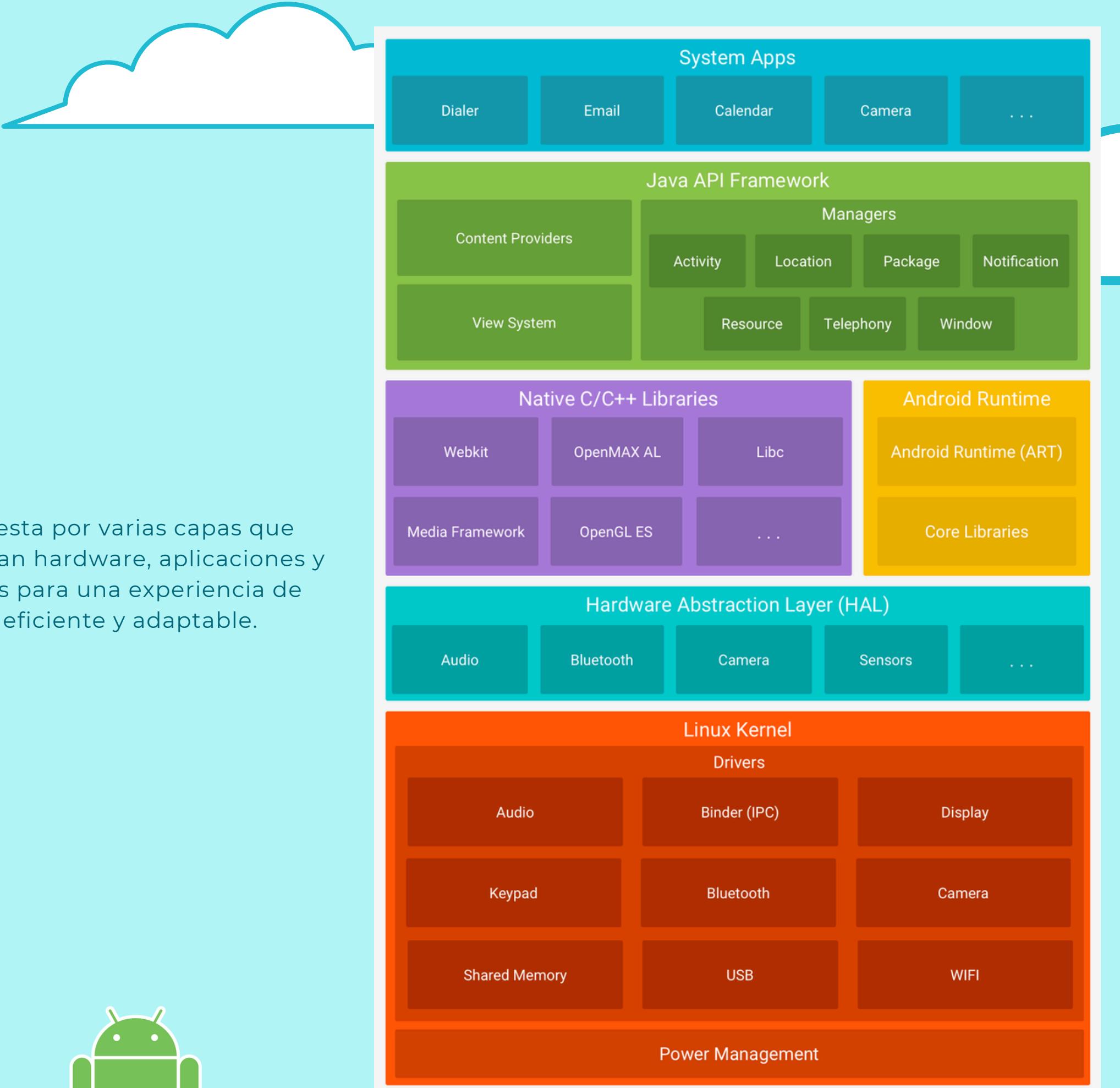
# Historia



El HTC Dream, lanzado en octubre de 2008, fue el primer teléfono Android.



Compuesta por varias capas que gestionan hardware, aplicaciones y servicios para una experiencia de usuario eficiente y adaptable.



# Ventajas

## Acceso Completo a Capacidades del Dispositivo

Aplicaciones nativas en Android tienen acceso completo a todas las funcionalidades del dispositivo, lo que permite una mayor interactividad y rendimiento.

## Rendimiento Robusto

Ofrece un rendimiento sólido tanto en modo online como offline, asegurando que las aplicaciones funcionen de manera efectiva en diversas condiciones.

## Amplia Variedad de Dispositivos

Está presente en una amplia gama de dispositivos, desde modelos económicos hasta premium, lo que permite a los usuarios elegir según sus necesidades y presupuesto.

# Desventajas

## Fragmentación del Sistema Operativo

Diversidad de dispositivos y personalizaciones de software puede generar experiencias desiguales

## Costos de Desarrollo

desarrolladores especializados en plataformas nativas pueden ser costosos, y se requiere una implementación diferente para cada plataforma, lo que limita la reutilización de código.

## Curva de Aprendizaje

Opción más costosa implica la necesidad de aprender nuevas herramientas y lenguajes, lo que puede ser un obstáculo para los nuevos desarrolladores.



Aplicaciones Fintech han crecido notablemente, utilizando innovaciones como la tecnología blockchain para ofrecer soluciones financieras. Ejemplos destacados incluyen Wise, PayPal, Revolut y Nubank, facilitando el acceso a productos y servicios financieros.

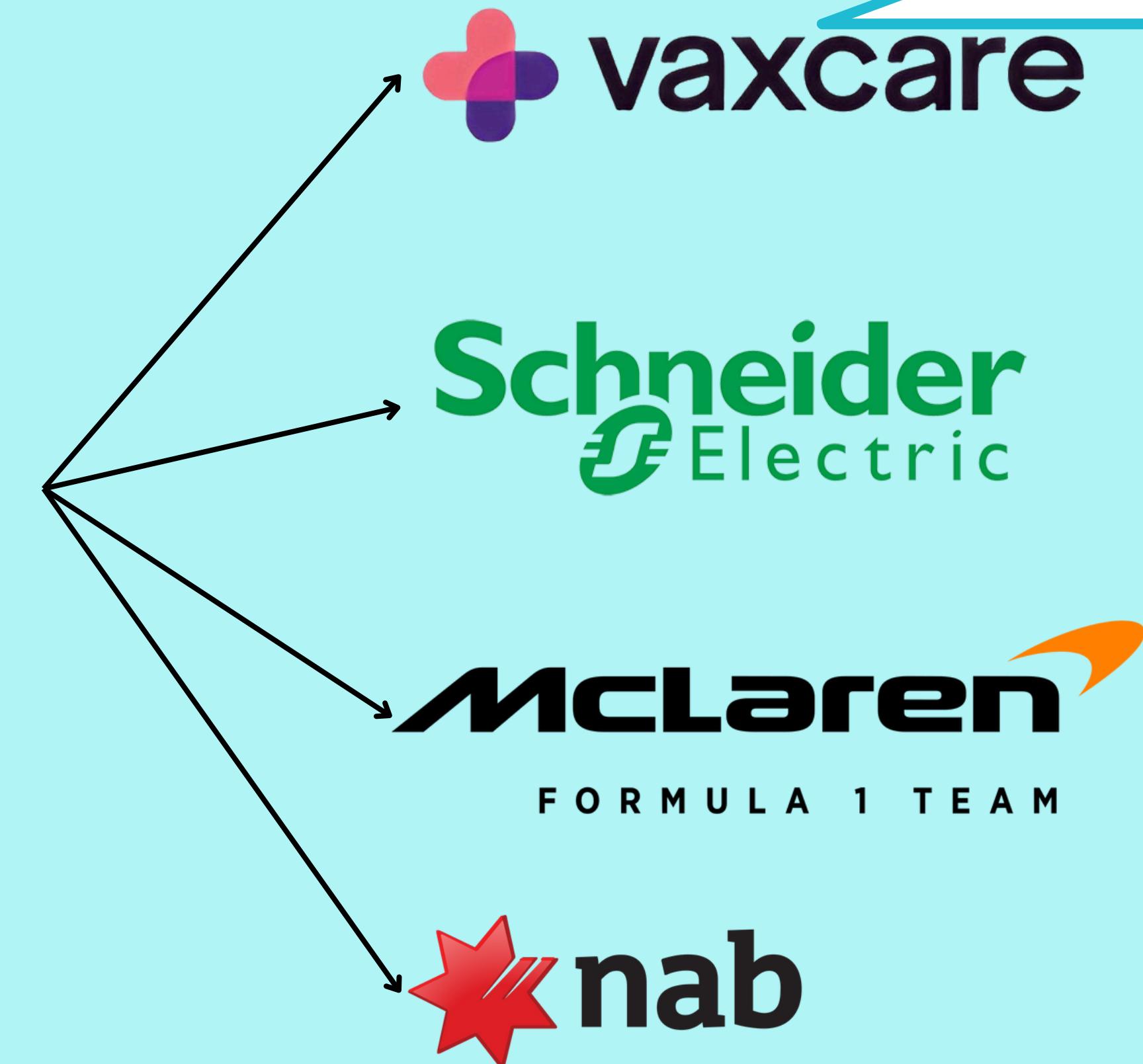


## Casos de uso

Aplicaciones de redes sociales como X, Facebook, TikTok e Instagram generan un alto tiempo de retención y crean grandes comunidades, influyendo significativamente en la interacción y comunicación de las personas en la sociedad.

Las aplicaciones de estilo de vida, como Uber y Spotify, y las de productividad, como Slack y Trello, tienen un gran potencial en el mercado, mejorando la vida diaria de los usuarios y aumentando la eficiencia en entornos laborales.

## Casos de aplicación



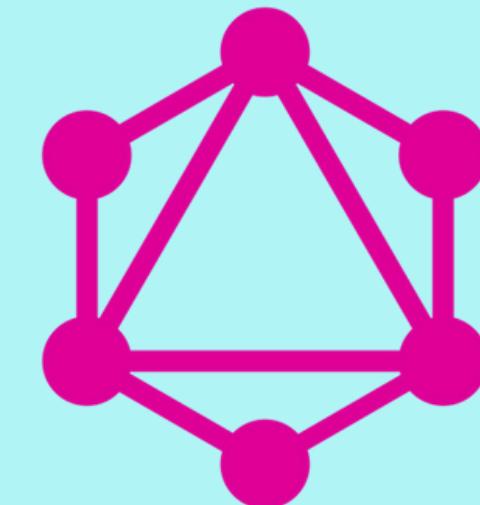
# Definición

- Lenguaje de consulta para APIs que transforma la manera en que los desarrolladores interactúan con los datos.
- Diseñado para ofrecer mayor flexibilidad y eficiencia que las arquitecturas REST tradicionales.
- Permite a los clientes solicitar solo la información necesitan, optimizando así el uso de recursos y mejorando la experiencia del usuario.

Consulta Precisa  
de Datos

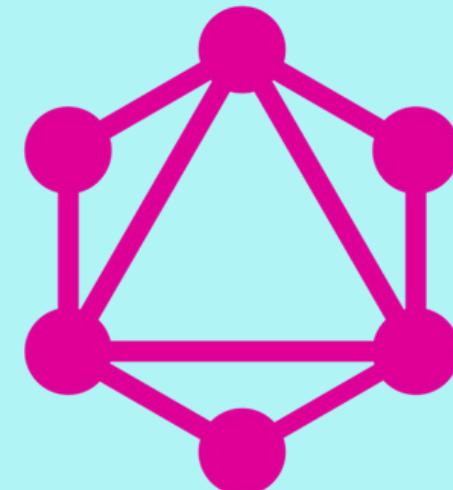
Sistema de Tipos

Único Endpoint



# GraphQL

# Características



## GraphQL

1

Utiliza un esquema estrictamente tipado que describe los datos disponibles, lo que permite a los clientes validar y prever errores antes de ejecutar consultas.

2

Permite definir la estructura exacta de las respuestas, evitando el "over-fetching" y "under-fetching" de datos.

3

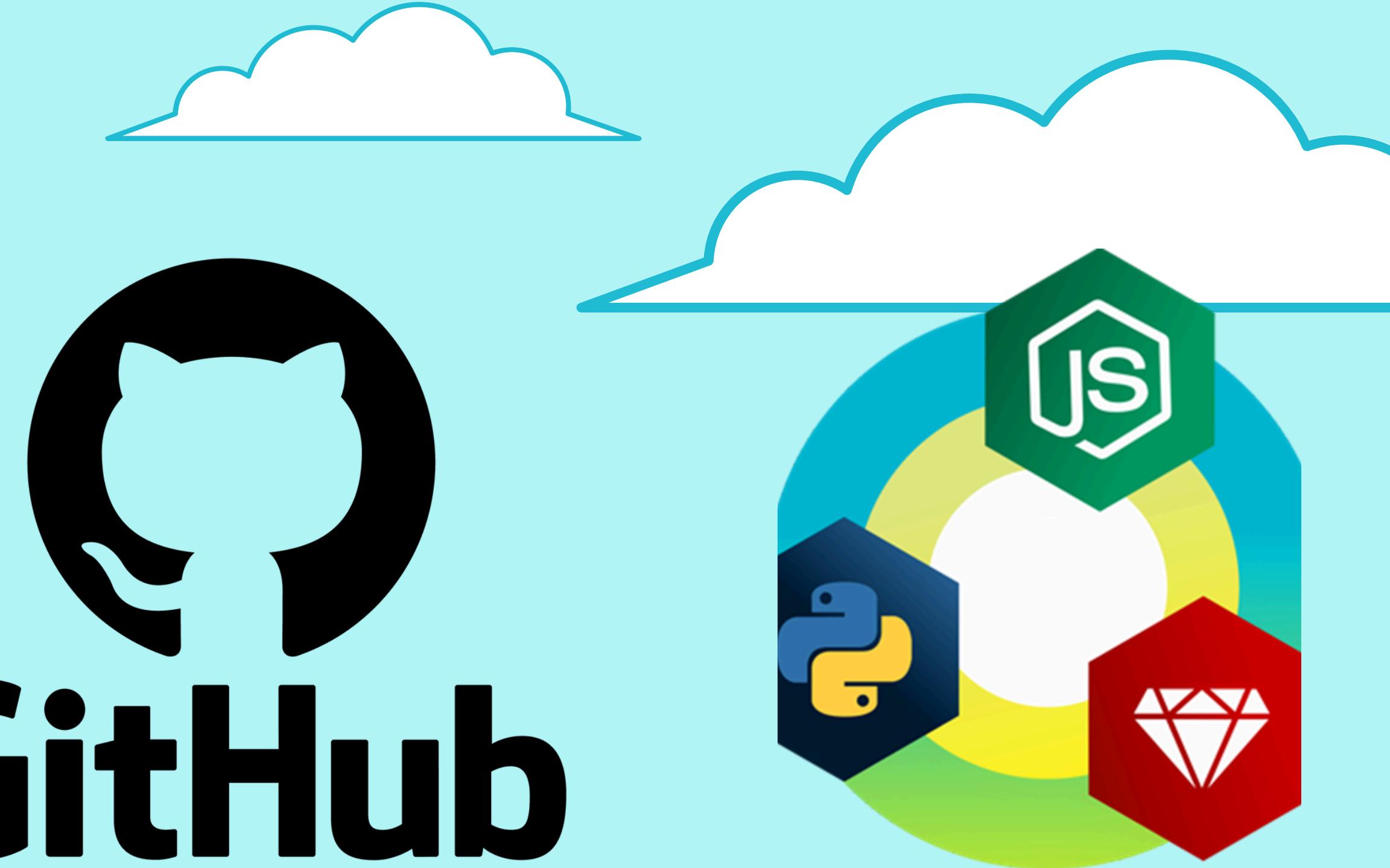
Facilita la eliminación gradual de campos obsoletos mediante deprecación, permitiendo que las APIs evolucionen sin afectar a los clientes existentes.

## Historia

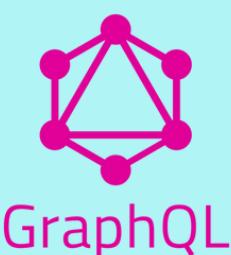


Desarrollo Inicial por Facebook, diseñado para solucionar problemas específicos de su app móvil, enfocándose en optimizar el tráfico de red y mejorar la eficiencia en las consultas.

## GitHub



Desde su publicación en 2015, empresas como GitHub, Twitter, Shopify y Airbnb han implementado GraphQL, destacando su flexibilidad y capacidad para manejar consultas complejas.



GraphQL

Ha evolucionado para convertirse en un estándar para APIs, con implementaciones en múltiples lenguajes y frameworks, mejorando la entrega de datos y la experiencia del usuario.

# Ventajas

## Optimización de datos

Al permitir que los clientes soliciten solo los datos que necesitan, se reduce significativamente el tráfico de red, lo que mejora la eficiencia en aplicaciones móviles y de baja conectividad.

## API flexible y evolutiva

Permite agregar nuevos campos al esquema sin afectar a los clientes existentes, lo que evita la necesidad de mantener múltiples versiones de la API.

## Mayor coherencia entre cliente y servidor

Al ser fuertemente tipada de GraphQL facilita la generación de documentación automática y la validación de consultas.

# Desventajas

## Consultas complejas pueden sobrecargar el servidor

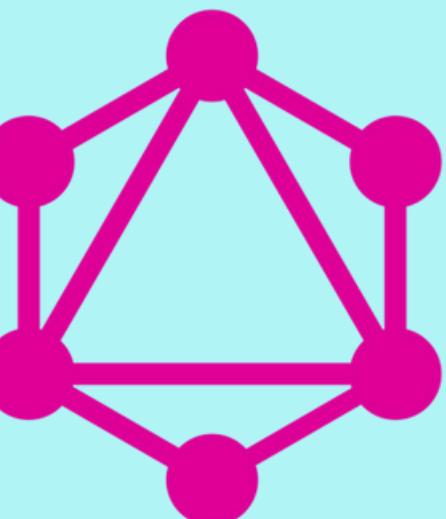
Las consultas anidadas o muy complejas pueden resultar en una mayor carga del servidor si no se gestionan correctamente.

## Mayor curva de aprendizaje

Los equipos acostumbrados a trabajar con REST pueden necesitar tiempo para aprender a modelar los datos y las consultas de manera eficiente en GraphQL.

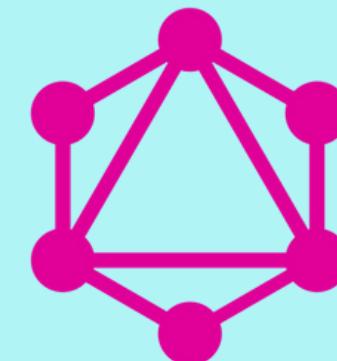
## Falta de control granular en caché

Aunque permite obtener datos específicos, carece de un mecanismo de caché nativo como REST, lo que puede aumentar la carga del servidor en algunas situaciones.



# GraphQL

Permite que los clientes móviles reciban solo los datos necesarios, lo que reduce el uso de ancho de banda.



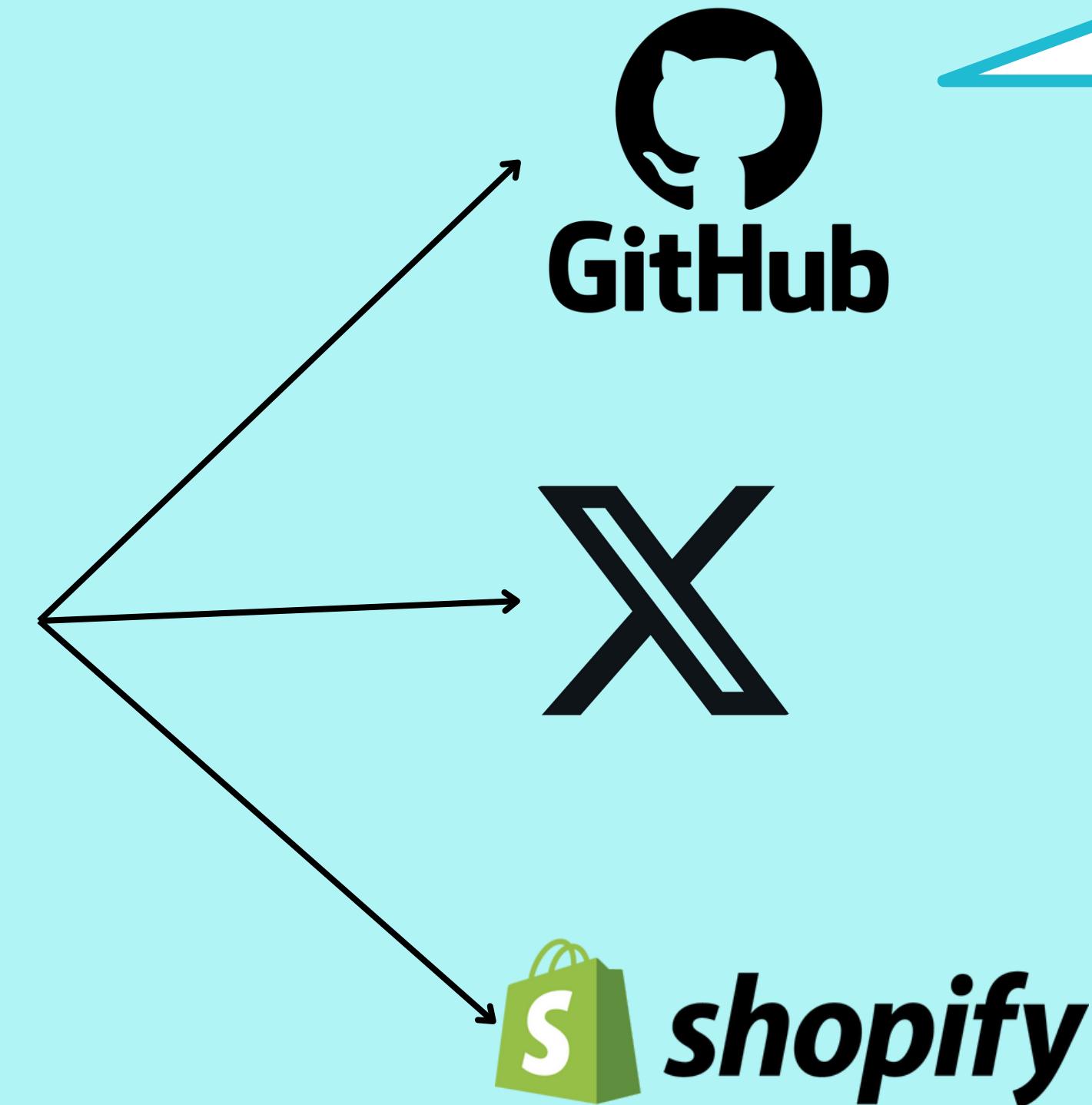
GraphQL

## Casos de uso

En aplicaciones donde múltiples componentes necesitan diferentes tipos de datos, GraphQL permite satisfacer todas estas necesidades en una única consulta, mejorando el rendimiento y reduciendo la complejidad del código del cliente.

Al operar sobre un único endpoint, GraphQL puede unificar datos de varios microservicios en una sola respuesta. Esto simplifica las interacciones entre sistemas distribuidos y mejora el rendimiento general.

## Casos de aplicación



## Definición

Ruby on Rails (RoR) es un potente framework de desarrollo web de código abierto que utiliza el lenguaje Ruby. Su enfoque en la simplicidad y la eficiencia facilita la creación de aplicaciones web, permitiendo a los desarrolladores concentrarse en la lógica empresarial en lugar de tareas repetitivas.



**Estructura MVC**

**Automatización de tareas**

**Versatilidad**

# Características



1

Permite a los desarrolladores seguir convenciones estándar, lo que reduce la necesidad de configuraciones extensivas y acelera el proceso de desarrollo.

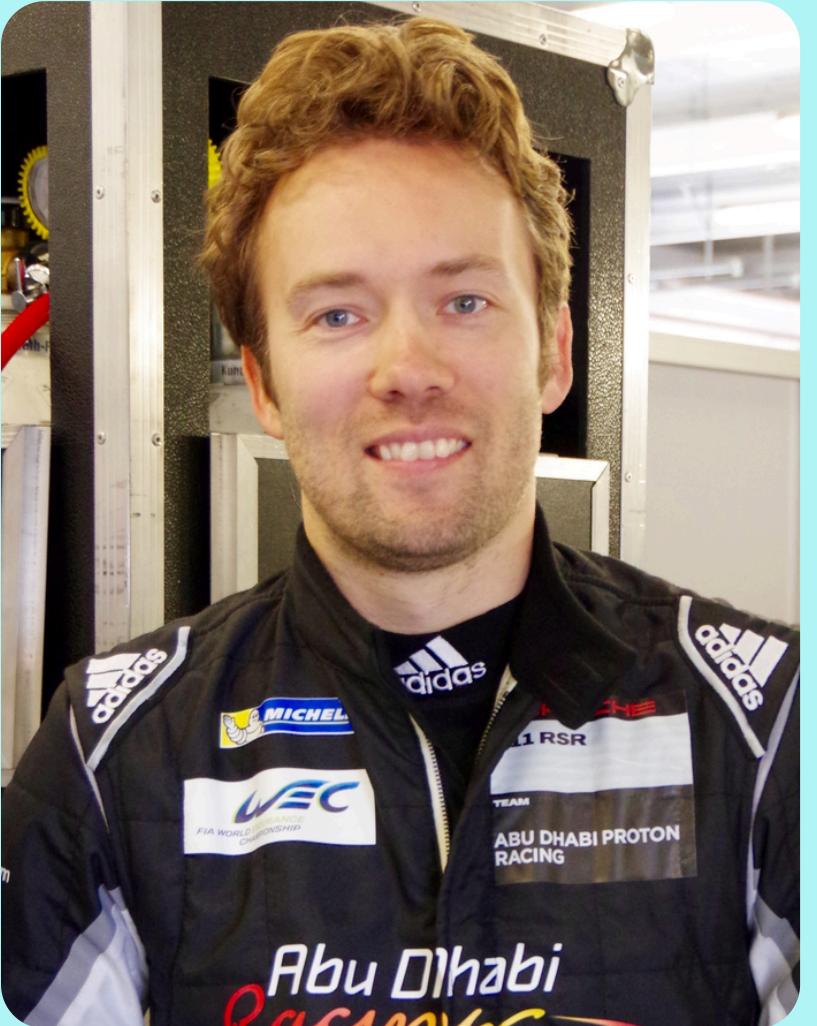
2

Fomenta la escritura de código limpio y reutilizable, minimizando la redundancia y mejorando la mantenibilidad del código.

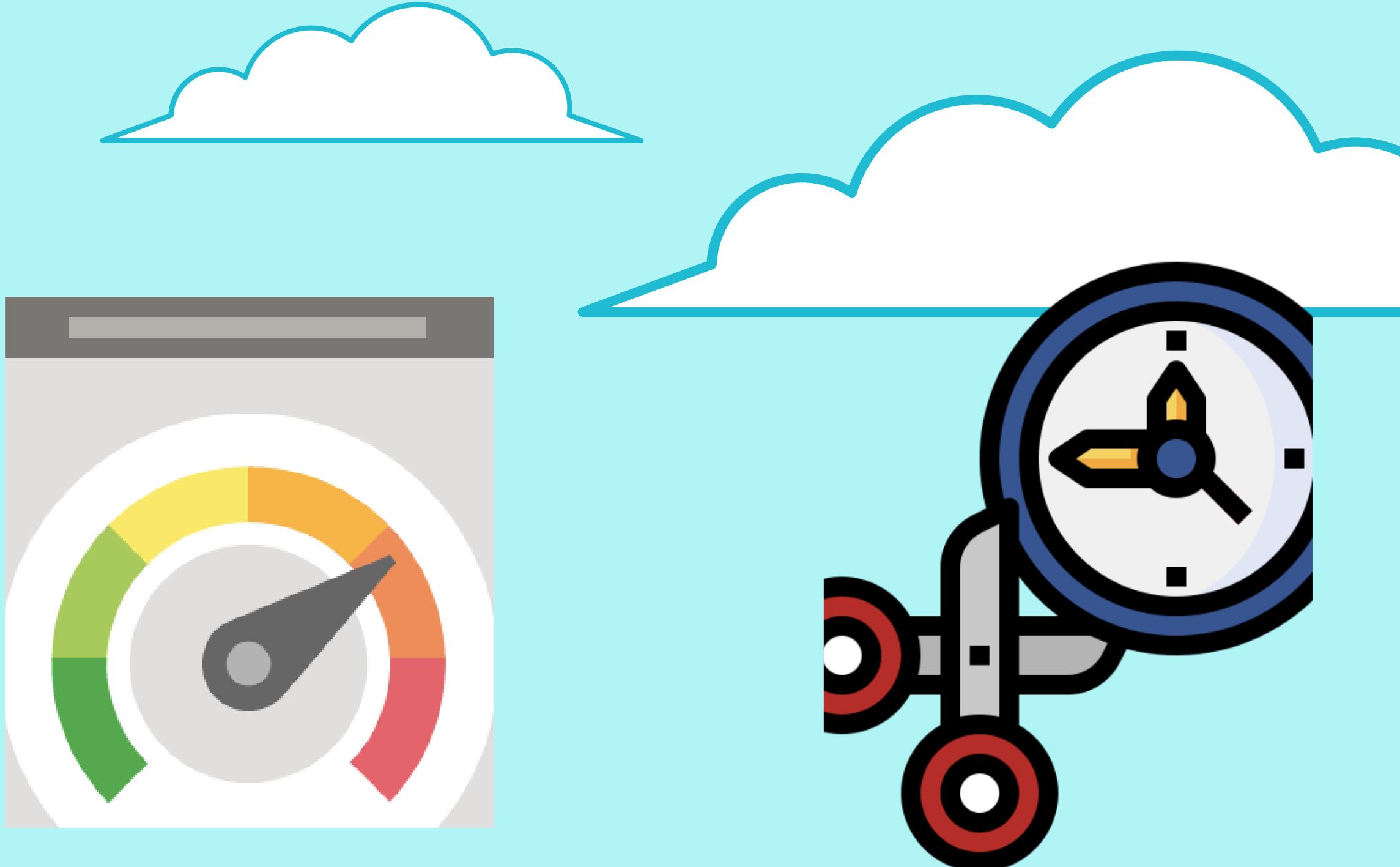
3

Incluye generadores que crean rápidamente estructuras comunes y permite la integración de múltiples bibliotecas, optimizando el desarrollo en diversos escenarios.

# Historia



Creado por David Heinemeier Hansson en 2003 y lanzado en 2004 como parte del proyecto Basecamp, con el objetivo de simplificar el desarrollo de aplicaciones web.



Rails ha evolucionado, mejorando su rendimiento, seguridad y capacidad de integración con tecnologías emergentes como Hotwire, que permite interactividad en tiempo real.

Ha ganado popularidad por su capacidad para acortar tiempos de desarrollo y su idoneidad para proyectos ágiles, convirtiéndose en una opción preferida para startups.



# Ventajas

## Rapidez en el desarrollo

Permite crear aplicaciones de forma rápida gracias a herramientas como generadores automáticos de código, lo que es ideal para startups y prototipos.

## Comunidad activa

Gran cantidad de recursos, bibliotecas (gems), y foros de discusión están disponibles, lo que mejora el soporte y la productividad.

## Convención sobre configuración

Simplifica el desarrollo siguiendo convenciones predefinidas que evitan la necesidad de configuraciones complejas.

# Desventajas

## Rendimiento inferior en aplicaciones de alta demanda

Comparado con otros lenguajes como Java o Node.js, Rails puede ser más lento en aplicaciones que manejan grandes volúmenes de tráfico o en tiempo real.

## Problemas de escalabilidad

Aunque es eficiente para proyectos pequeños y medianos, cuando las aplicaciones crecen en complejidad o tamaño, Rails puede requerir optimizaciones adicionales.

## Consumo de recursos

En servidores, Rails tiende a consumir más memoria y CPU en comparación con otros frameworks más ligeros.





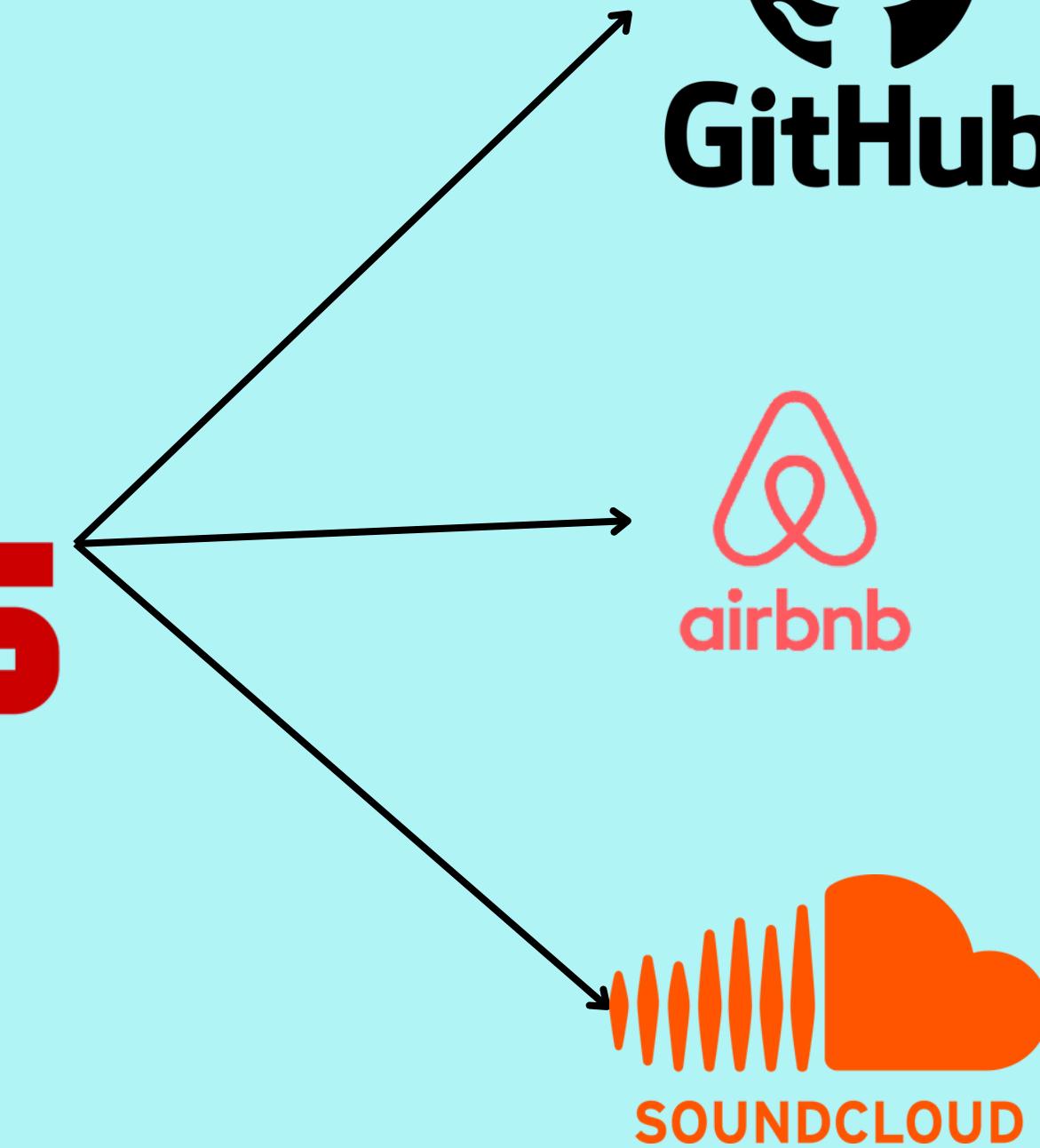
Ideal para aplicaciones que requieren una gestión eficiente de datos, como plataformas de comercio electrónico, donde se manejan usuarios, productos y transacciones.



Adecuado para sistemas de gestión de contenido (CMS) y redes sociales, donde las actualizaciones en tiempo real y la capacidad de manejar múltiples usuarios son esenciales.

Se utiliza ampliamente en proyectos de software colaborativo y en plataformas que necesitan integraciones con servicios de terceros.

## Casos de aplicación



## Definición

- Sistema de gestión de bases de datos de código abierto
- Diseñado para manejar grandes volúmenes de datos estructurados de manera **eficiente y escalable**.
- Destaca por su capacidad para operar en entornos de gran tamaño, ofreciendo alta disponibilidad y redundancia.

Escalabilidad

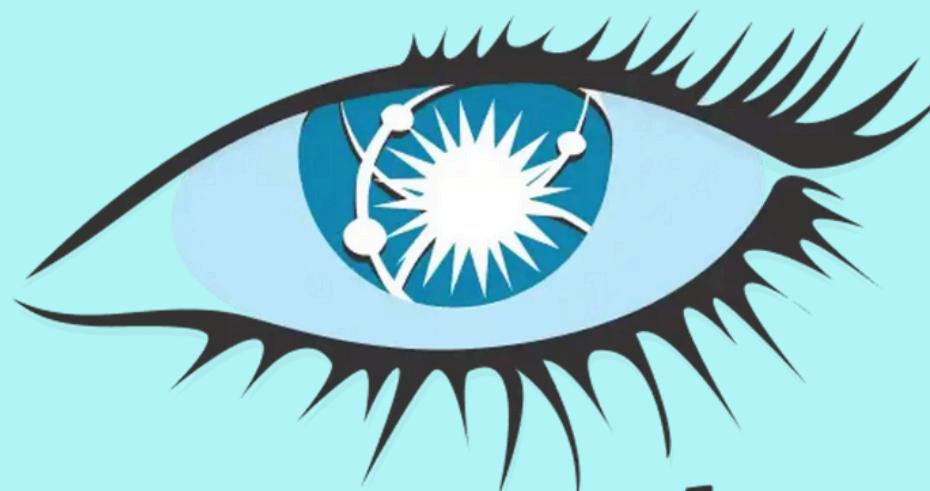
Redundancia y disponibilidad

Flexibilidad de despliegue



*cassandra*

# Características



**cassandra**

1

Sistema distribuido, donde los servidores están distribuidos entre múltiples nodos. Ofrece escalabilidad horizontal y lineal, lo que significa que al duplicar el número de nodos

2

Tolerante a fallos gracias a su sistema de replicación de datos. Si un nodo falla, los datos ya estarán replicados en otros nodos, garantizando su disponibilidad.

3

Permite la replicación de datos en múltiples centros de datos, creando "anillos" de máquinas Cassandra, donde los datos del anillo 1 pueden replicarse en el anillo 2.

# Historia



El nombre esta inspirado en Cassandra, la sacerdotisa de la mitología griega.



**amazon**  
DynamoDB



**Google**  
BigTable



Inspirada en Amazon Dynamo y Google BigTable, que sentaron las bases para el manejo de grandes volúmenes de datos distribuidos.



- 2008 → Lanzada por Facebook para mejorar la búsqueda en su sistema de mensajes.
- 2010 → Transferida a la Fundación Apache, donde se convirtió en un proyecto de nivel superior.
- Actualmente → Es liderado por DataStax

# Ventajas

## Alta disponibilidad

Ideal para sistemas críticos donde las caídas no son tolerables, ya que está diseñada para funcionar con mínima interrupción.

## Tolerancia a particiones y escalabilidad

Soporta la tolerancia a particiones, lo que le permite seguir operando incluso cuando hay problemas de red o fallos de nodos. Además, es altamente escalable, pudiendo crecer sin afectar su rendimiento.

## Distribución de datos flexible

Permite distribuir datos fácilmente a través de múltiples centros de datos, lo que facilita su uso en diferentes ubicaciones y asegura la redundancia geográfica.

# Desventajas

## Complejidad en la adición de nodos

El sistema debe sincronizarse con los nodos existentes.

## Optimización de consultas

Requiere que las consultas (queries) estén bien definidas con antelación, ya que sufre en rendimiento al realizar consultas tipo "SELECT" debido a la forma en que los datos están almacenados.

## Mayor complejidad operativa

Administración y optimización más complicada en comparación con otros, requiriendo un conocimiento profundo para su correcta configuración y mantenimiento.



Las aplicaciones que requieren generar recomendaciones personalizadas (por ejemplo, Netflix o Amazon) pueden beneficiarse de Cassandra. Este tipo de sistemas necesitan almacenar grandes volúmenes de datos de usuarios y comportamientos

Los sitios web de comercio electrónico requieren manejar grandes cantidades de datos sobre productos, clientes, y transacciones. Cassandra permite gestionar catálogos de productos en diferentes regiones geográficas

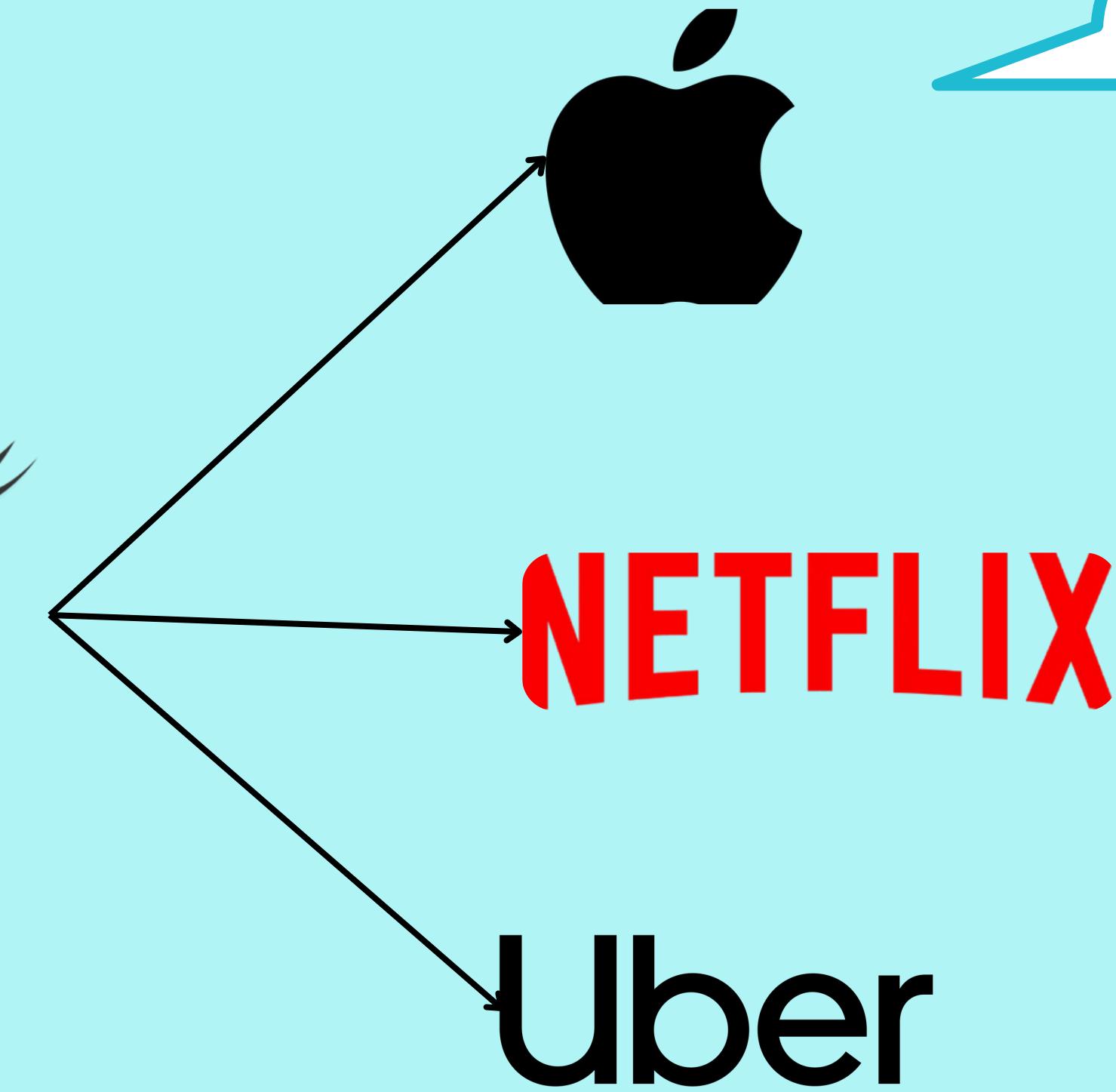


Empresas que necesitan almacenar registros detallados de eventos o actividades, como logs de servidores, transacciones bancarias o actividades de usuarios en aplicaciones web, utilizan Cassandra debido a su capacidad para escribir grandes volúmenes de datos de manera rápida y eficiente

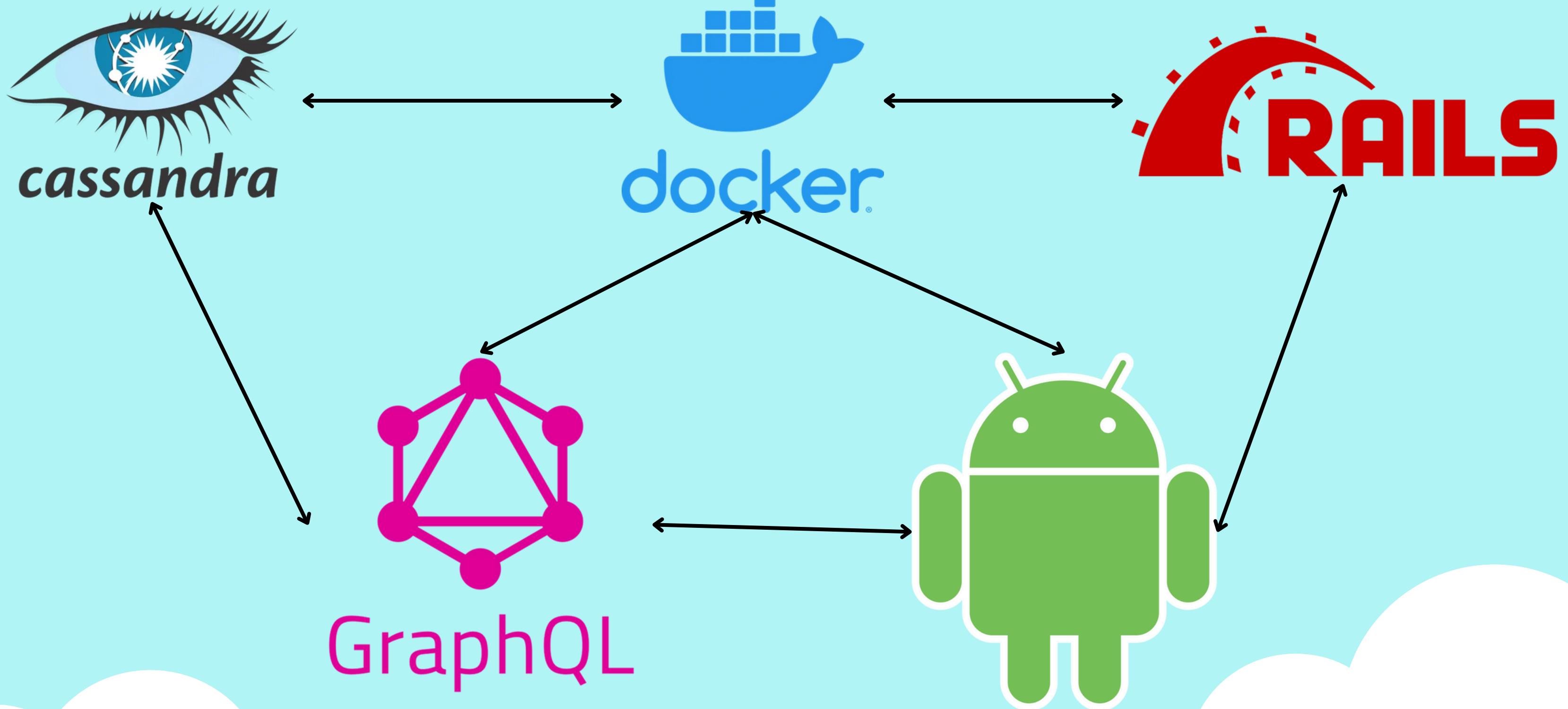
## Casos de aplicación



*cassandra*



# ¿Qué tan común es el stack designado?



# Principios SOLID vs Temas

	Android native	GraphQL	Ruby on rails	Cassandra	CQRS
Single Responsibility	Medio	Alto	Medio	Bajo	Alto
Open/Closed	Medio	Alto	Alto	Medio	Medio
Liskov Substitution	Bajo	Medio	Alto	Bajo	Medio
Interface Segregation	Alto	Medio	Medio	Bajo	Alto
Dependency Inversion	Medio	Medio	Alto	Bajo	Medio

# Atributos de calidad vs temas

	Android native	GraphQL	Ruby on rails	Cassandra	CQRS
Idoneidad funcional	Alto	Alto	Alto	Medio	Alto
Eficiencia	Medio	Alto	Medio	Alto	Alto
Compatibilidad	Bajo	Medio	Medio	Bajo	Alto
Usabilidad	Alto	Alto	Alto	Medio	Bajo
Confiabilidad	Alto	Medio	Medio	Alto	Alto
Seguridad	Alto	Medio	Alto	Alto	Alto
Mantenibilidad	Medio	Alto	Alto	Medio	Medio
Portabilidad	Bajo	Alto	Medio	Medio	Alto

# Análisis de tácticas vs temas

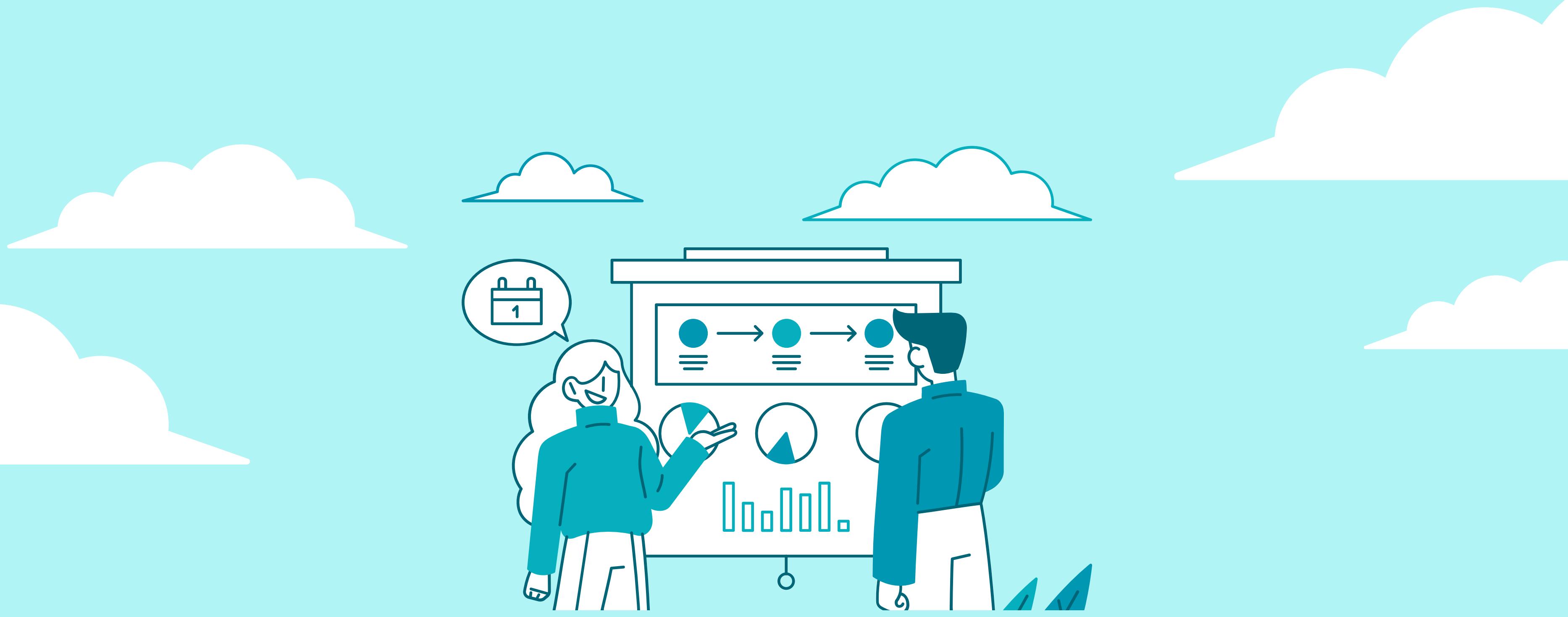
	Cassandra	RubyOnRails	GraphQL	Android
Optimización de consultas	Alto	Medio	Alto	Bajo
Manejo de concurrencia	Medio	Bajo	Alto	Medio
Manejo de errores y fallos	Medio	Alto	Medio	Medio
Escalabilidad	Alto	Medio	Alto	Bajo
Facilidad de integración	Bajo	Alto	Alto	Medio
Seguridad	Medio	Alto	Alto	Bajo
Replicación	Alto	Bajo	N/A	N/A
Monitoreo	Medio	Alto	Medio	Bajo

# Mercado laboral vs temas

	Android native	GraphQL	Ruby on rails	Cassandra	CQRS
Demanda	Alta	Alta	Media	Media	Alta
Nuevas ofertas	Media	Alta	Baja	Media	Media
Competencia	Alta	Alta	Alta	Media	Media
Necesidad	Alta	Alta	Media	Alta	Alta

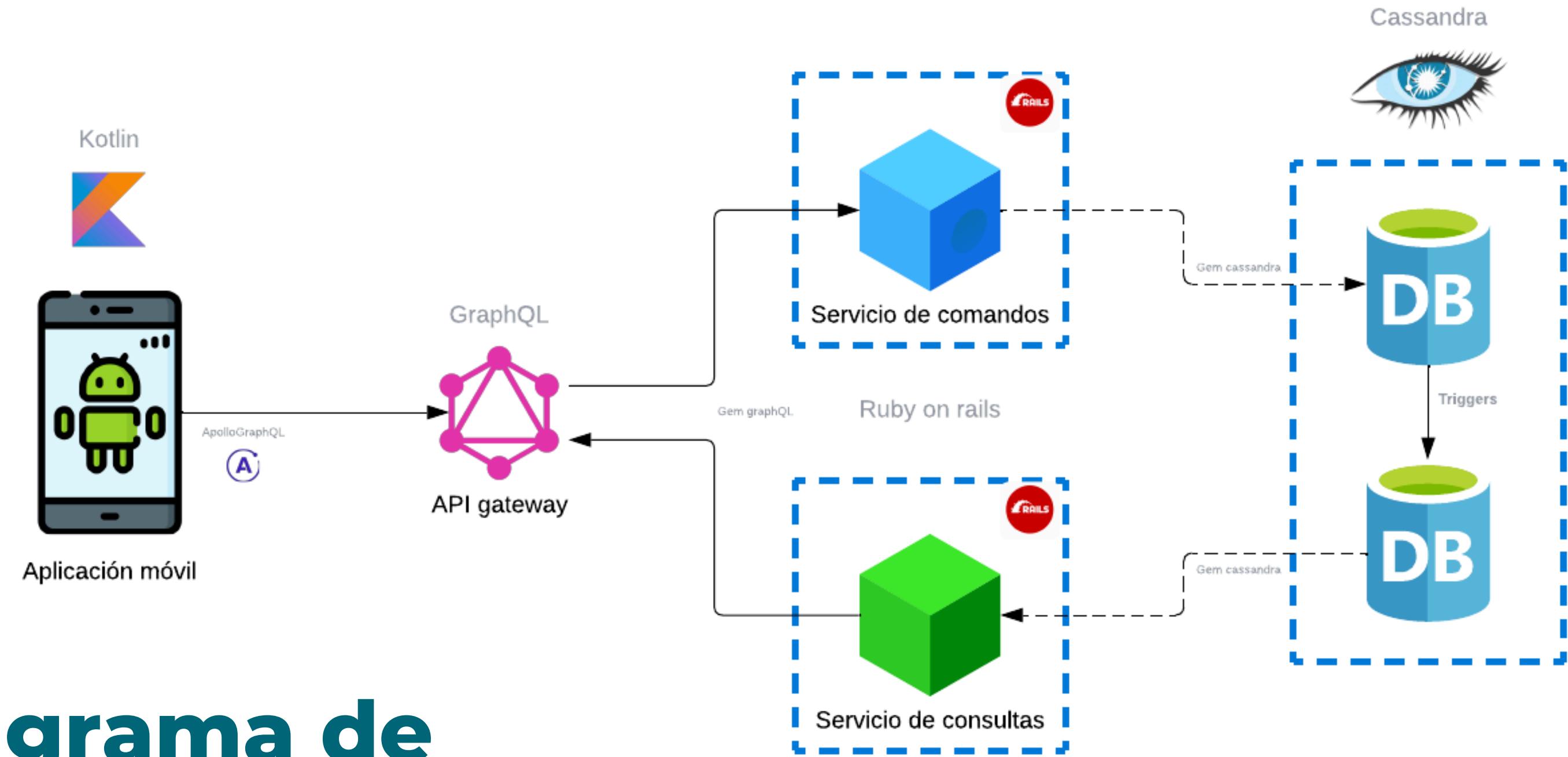
# Patrones laboral vs temas

	Cassandra	RubyOnRails	GraphQL	Android
Desarrollo ágil	Permite iteraciones rápidas en ajustes de nodos y consultas	Facilita la implementación y cambios rápidos en las APIs	Soporta cambios rápidos en resolvers y consultas	Facilita la iteración continua en el diseño y funcionalidad
Trabajo remoto	Uso de clústeres distribuidos que facilitan el acceso remoto a datos	Permite un despliegue flexible y remoto de aplicaciones	Permite mantener la eficiencia en la comunicación entre equipos distribuidos	Herramientas de Android Studio para desarrollo remoto
Colaboración en equipo	Replicación y sincronización de datos entre equipos de desarrollo	Ruby on Rails permite la integración con equipos de frontend a través de APIs estandarizadas	Se centraliza la comunicación entre equipos, facilitando el control de versiones de APIs	Uso de herramientas como Figma o Zeplin para sincronizar el diseño entre equipos
Mejora continua	Optimización constante de índices y queries para mejorar el rendimiento	Facilita la refactorización y mejora continua del código	Permite la actualización incremental de esquemas sin afectar al cliente	Adaptación continua de interfaces de usuario a nuevos estándares y dispositivos
Metologías ágiles	Gestión iterativa de nodos y consultas, permitiendo cambios rápidos y pruebas de concepto	Se adapta bien a Scrum y otras metodologías ágiles por su facilidad de desarrollo rápido	Es flexible para soportar cambios rápidos y ajustables en los requerimientos del cliente	Soporta iteraciones ágiles para probar rápidamente nuevas funcionalidades

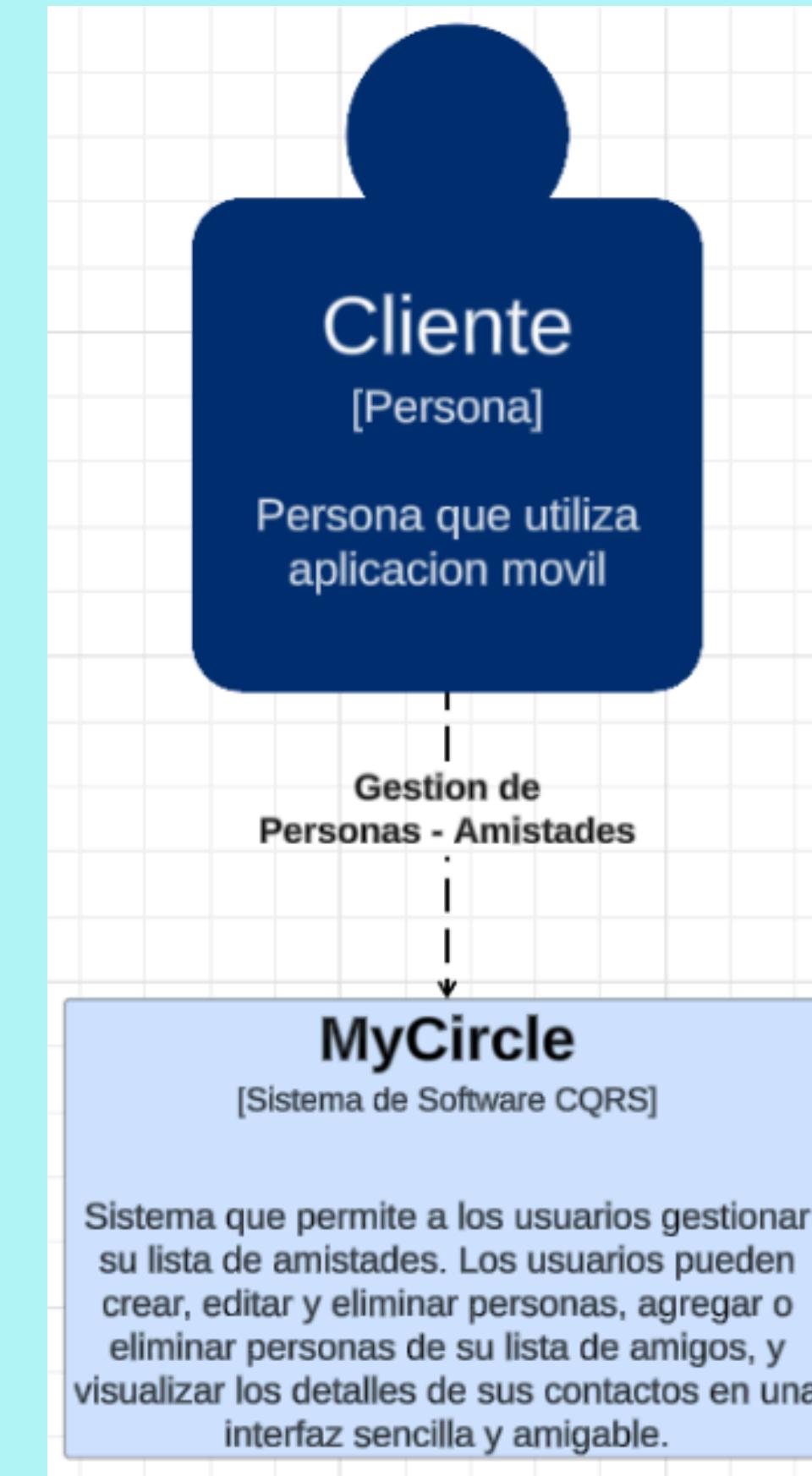


# Diagramas

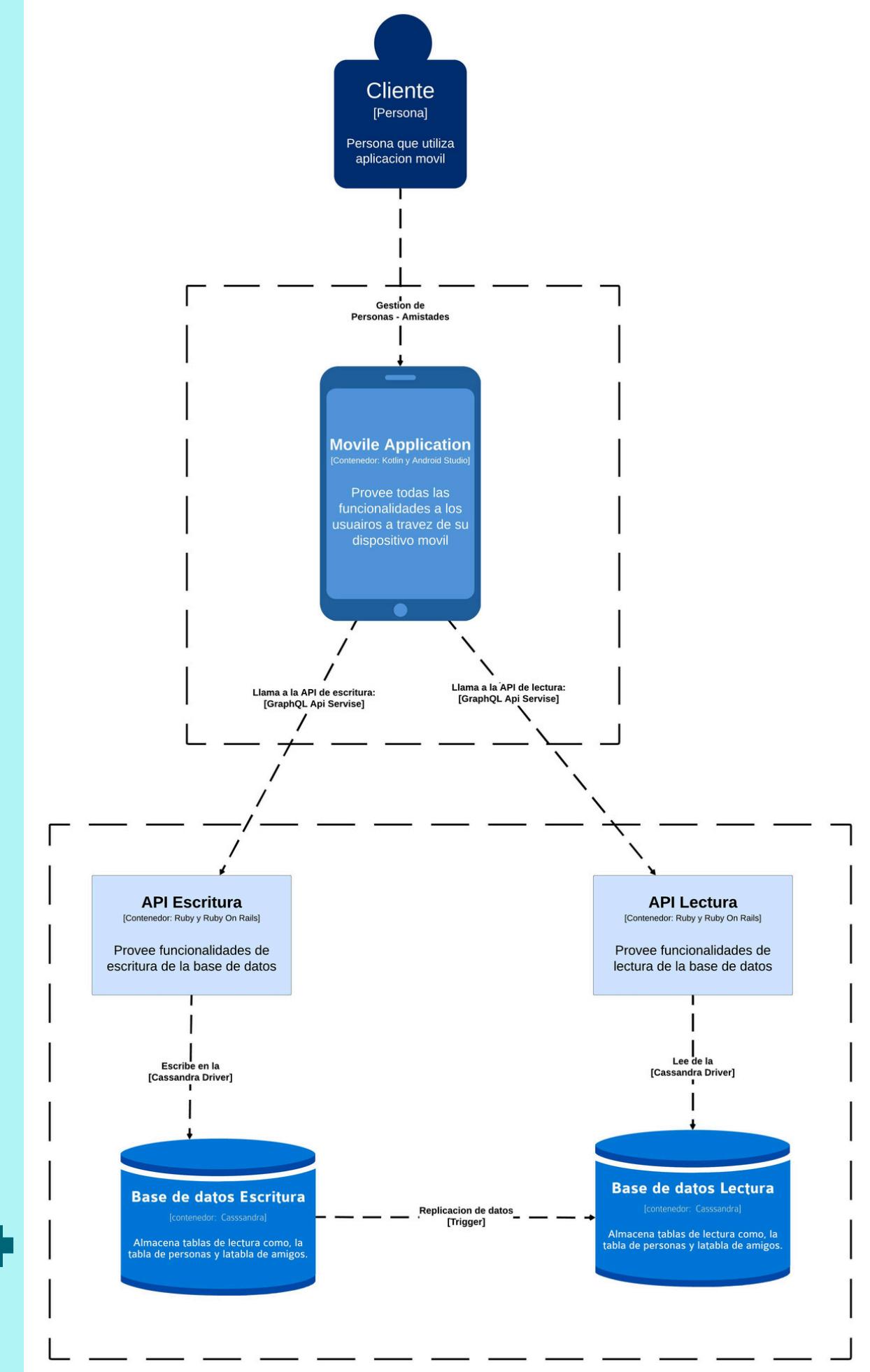
# Diagrama de arquitectura



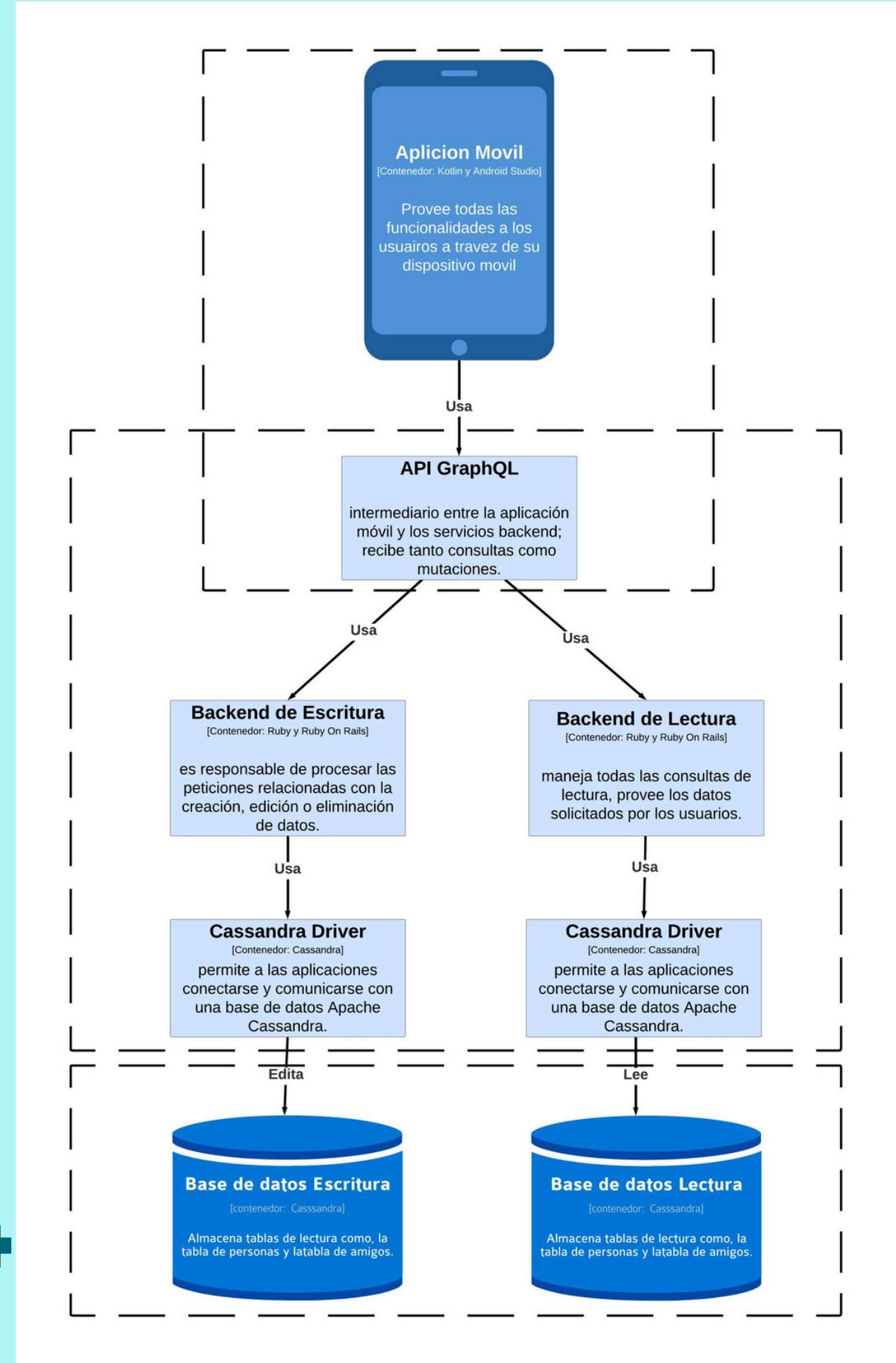
# Diagrama de Contexto C4

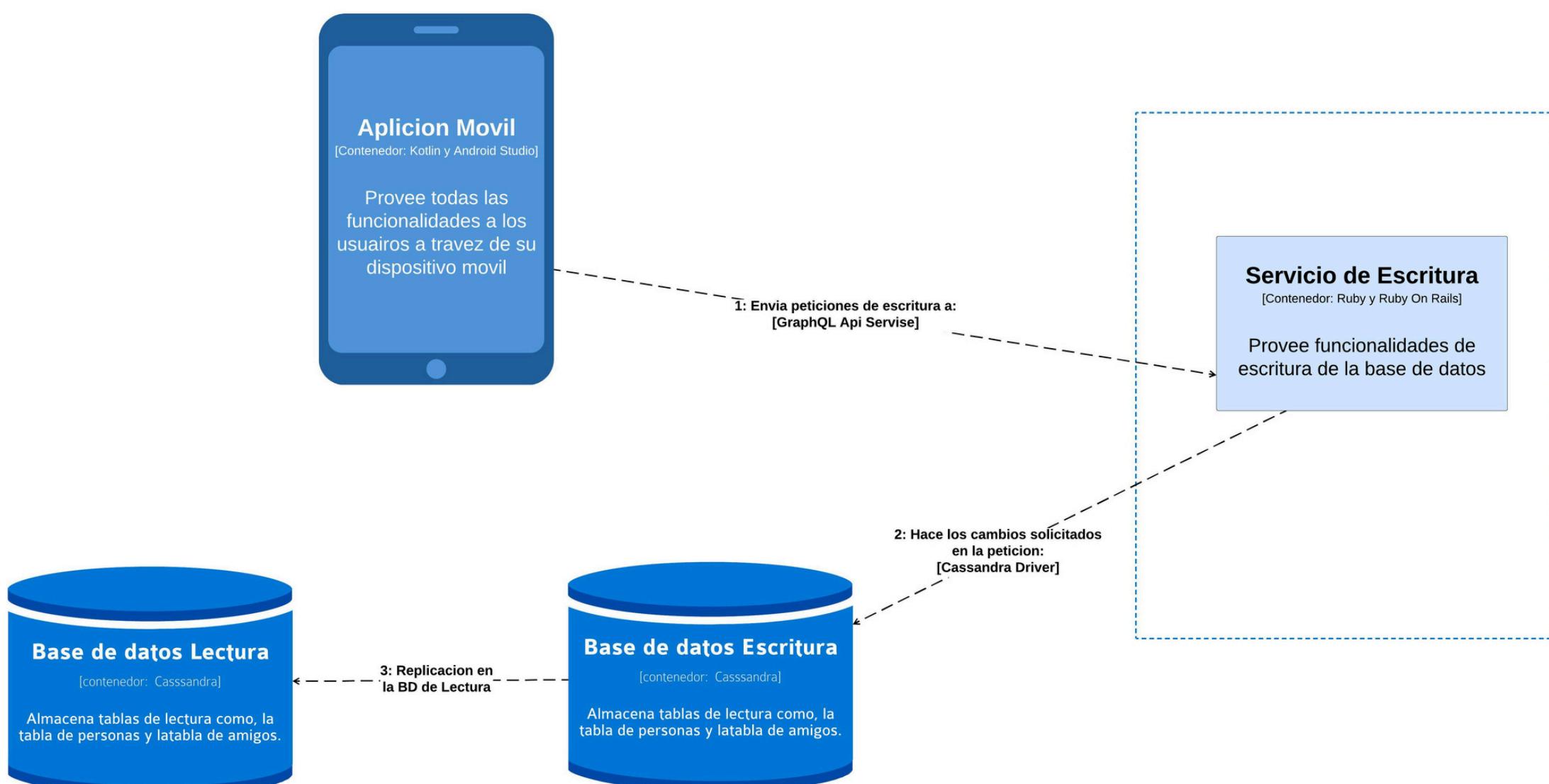
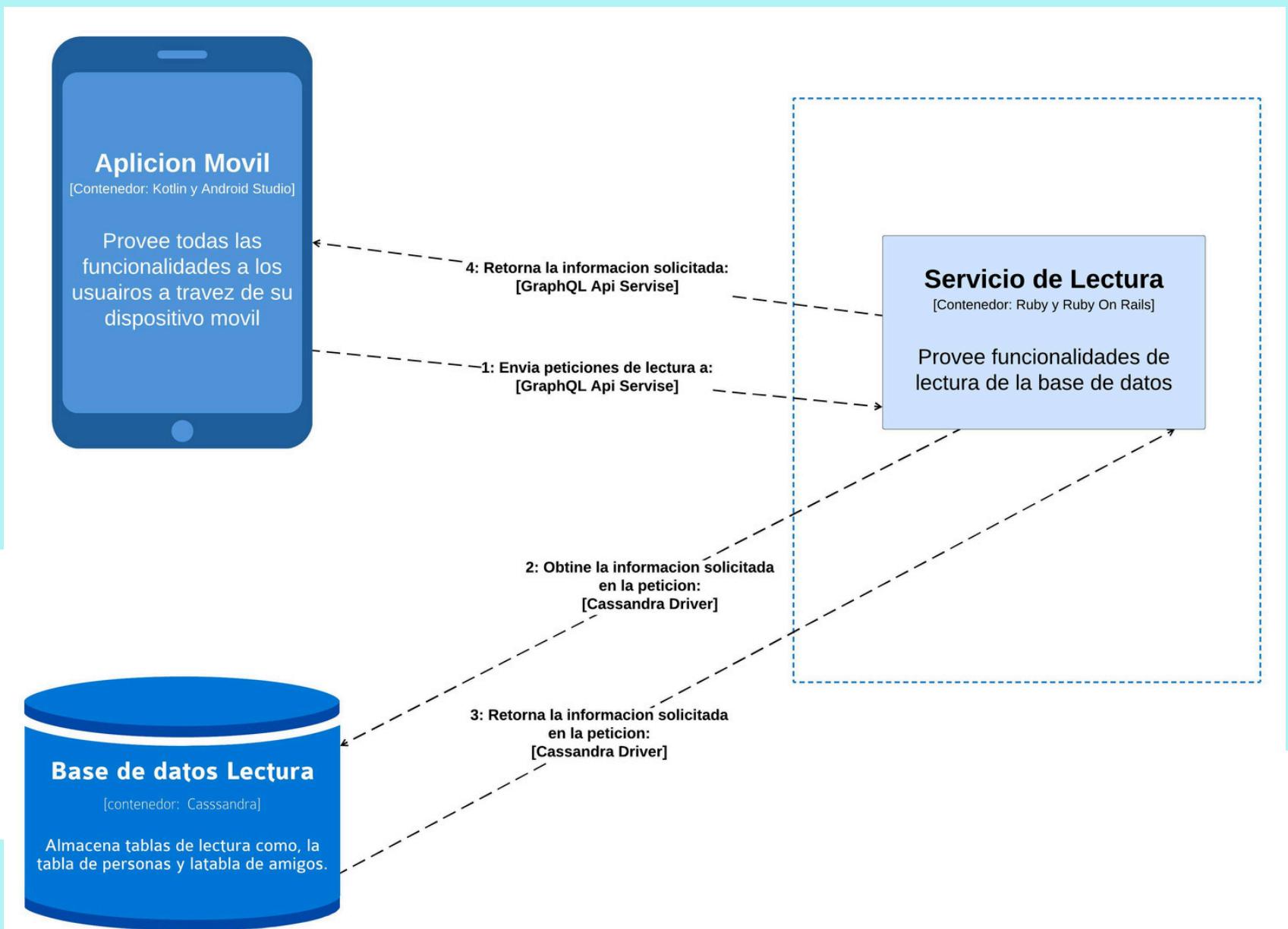


# Diagrama de Contenedores C4



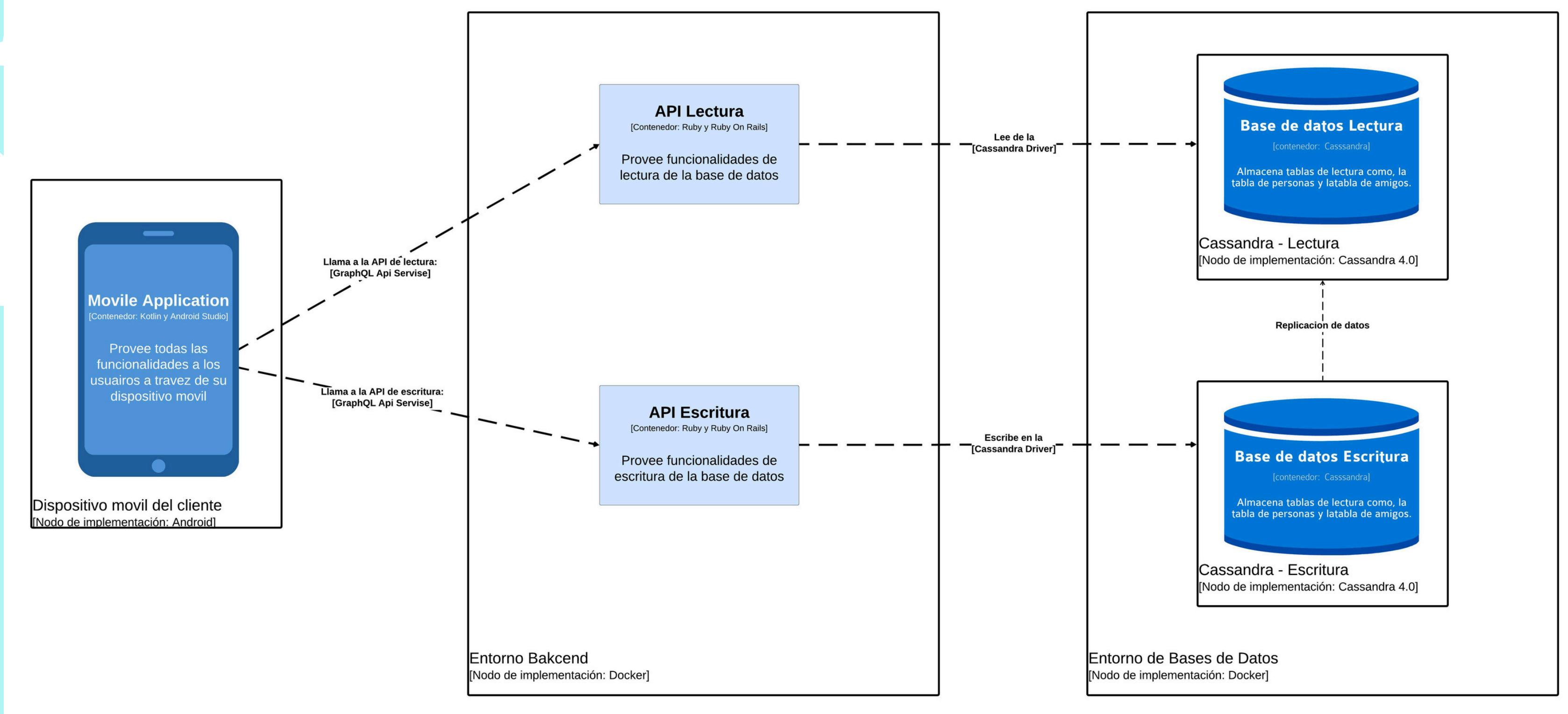
# Diagrama de Componentes C4





# Diagramas de Dinámicos C4

# Diagrama de Despliegue C4



# Diagrama UML de Paquetes

