GraphQL y gRPC

Análisis y diseño de aplicaciones I

Ingeniería en Informática

Integrantes del grupo:

- Cabrera, Matias
- Ferreira, Guillermo
- Glass, Carol
- Maidana, Juan
- Lorenzo, Agustin
- Rama, Tomas

29 de Junio 2023

Introducción

Tres tecnologías que pueden ser soluciones destacadas para una comunicación efectiva entre componentes de un sistema son GraphQL, gRPC y, la más conocida, REST.

GraphQL es un lenguaje de consulta y manipulación de datos que ofrece una forma flexible de interactuar con APIs. Permite definir consultas personalizadas que pueden obtener información de diferentes microservicios, filtrando los resultados para devolver sólo lo que se necesita, evitando over-fetching (sobrecarga) o under-fetching (falta) de datos. (Newman, 2021, 123). Con GraphQL, los desarrolladores pueden definir un esquema que describe los tipos de datos disponibles y las operaciones permitidas, lo que facilita la evolución de la API sin romper la compatibilidad con otras versiones. (*Schemas and Types*, n.d.)

Por otro lado, gRPC, es un framework de comunicación remota que se basa en el Protocolo de Buffers de Google (protobuf). Utiliza el paradigma de llamada a procedimientos remotos (Remote Procedure Call) y gRPC permite que los clientes realicen llamadas a métodos en servidores, como si estuvieran invocando métodos locales. (Newman, 2021, 123) Además, gracias a la serialización compacta de datos proporcionada por protobuf, gRPC ofrece una comunicación eficiente y de alto rendimiento. (*GraphQL Vs gRPC | GraphQL Tutorial*, n.d.)

En este documento, se explicarán ambas tecnologías y se presentarán las ventajas y desventajas que presentan. Se demostrará cómo gRPC puede ser implementada.

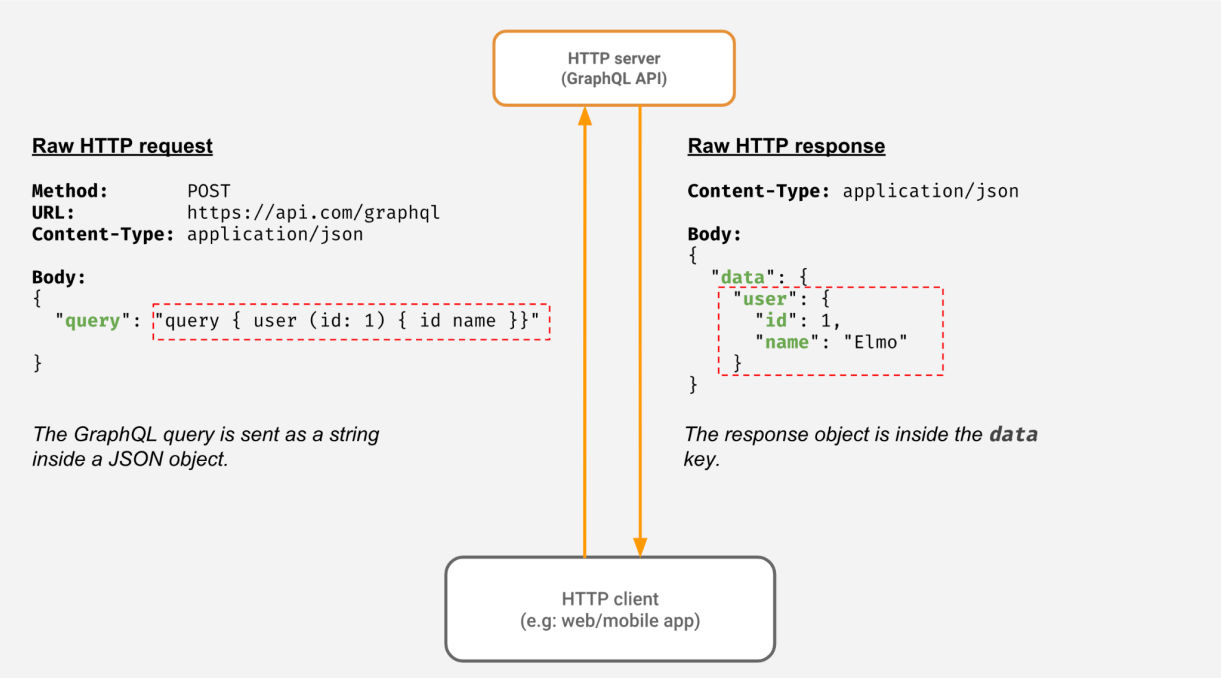
Desarrollo del problema

Aunque GraphQL y gRPC tienen propósitos similares, se basan en conceptos y enfoques diferentes.

GraphQL

GraphQL es un lenguaje de consulta y manipulación de datos desarrollado por Facebook y lanzado en Septiembre 2015, diseñado para que clientes HTTP puedan realizar llamadas a la API y obtener los datos que necesitan de sus API backend al proporcionar una capa de abstracción sobre los datos subyacentes . Por otro lado, también permite la mutación de datos, lo que permite modificar o crear nuevos datos en el servidor.

La siguiente imagen muestra un diagrama explicativo sobre cómo GraphQL es utilizado en la comunicación entre el HTTP Client y HTTP Server.



Flujo GraphQL cliente-servidor (*What Is GraphQL? | GraphQL Tutorial*, n.d.)

1. La consulta GraphQL se parece a la forma del JSON. Por lo tanto, en lugar de realizar un GET a cada URL como para REST, se hacen POST enviando la consulta al servidor, éstas son enviadas como "string" por el cliente.
2. El servidor obtiene el objeto JSON y extrae la cadena de consulta. Según la sintaxis GraphQL y el esquema, el servidor procesa y valida la consulta.
3. Al igual que otros servidores API, el de GraphQL realiza llamadas a la base de datos u otros servicios para obtener los datos solicitados por el cliente.
4. Al final, el servidor toma los datos y los devuelve al cliente en un objeto JSON.

```
Request1:          | Response1:

query {            | {
  user (id: 1) {   |   "user": {
    id             |     "id": 1
  }                |   }
}                  | }

-----------------------------------

Request2:          |  Response2:

query {            |  {
  user (id: 1) {   |    "user": {
    id             |      "id": 1
    name           |      "name": "Elmo"
  }                |    }
}                  |  }
```

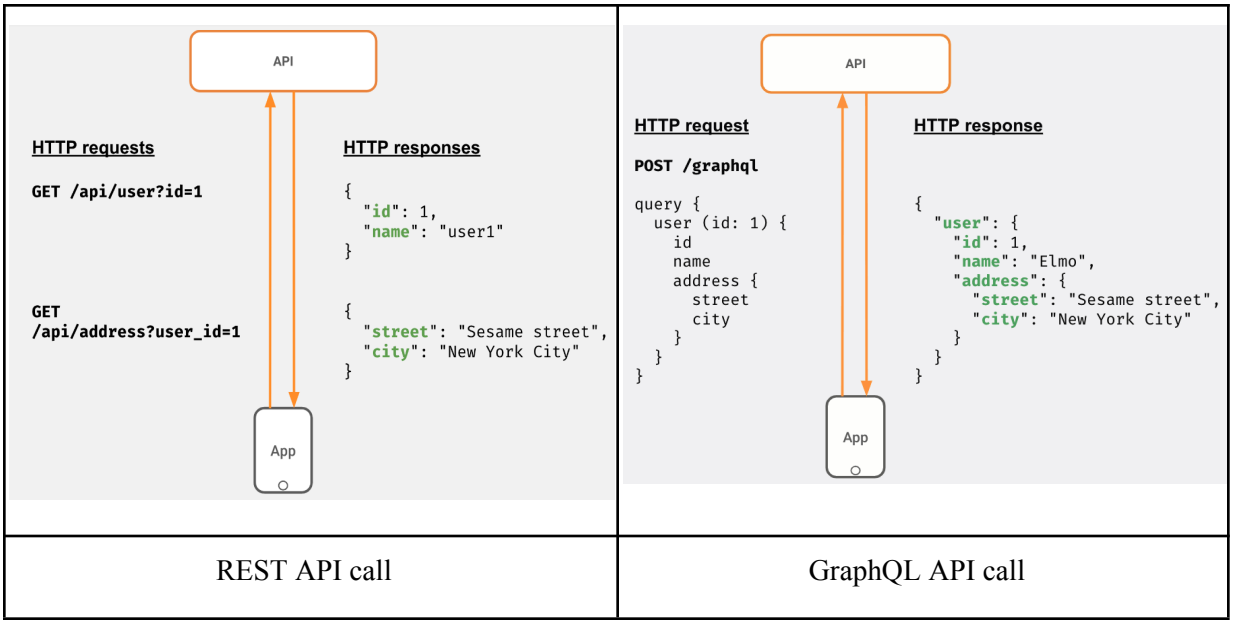
Manejo de HTTP Status Codes:

Cada solicitud GraphQL, éxito o error debe devolver un 200, es decir, cualquier respuesta es válida y los errores se manejan como parte del cuerpo. Esta es una gran diferencia en comparación con las API REST, donde cada código de estado apunta a un determinado tipo de respuesta. (*GraphQL Vs REST | GraphQL Tutorial*, n.d.).

En cuanto a las ventajas y desventajas de GraphQL, algunas de ellas son:

Ventajas de GraphQL:

1. Evolución del API: El esquema GraphQL permite realizar cambios sin romper la compatibilidad con versiones anteriores.
2. Rendimiento optimizado: Al enviar sólo los datos necesarios, se reduce el tamaño de la respuesta mejorando la eficiencia de las comunicaciones. Evita múltiples llamadas a la API. En la siguiente comparación, para el caso de GraphQL no sería necesario realizar 2 llamadas a la API para obtener el usuario y la dirección por separado.



REST API call	GraphQL API call

3. Sencillez de lectura y entendimiento: pues, como se mencionó anteriormente, las consultas coinciden con la "forma" de los datos JSON finales que se desea.
4. Flexibilidad: Los datos devueltos son específicos según la consulta del cliente, esto permite que el servidor se adapte fácilmente sin afectar a los clientes que ya existen, evitando también problemas de compatibilidad. (*GraphQL Advantages and Disadvantages - Javatpoint*, n.d.)

Desventajas de GraphQL:

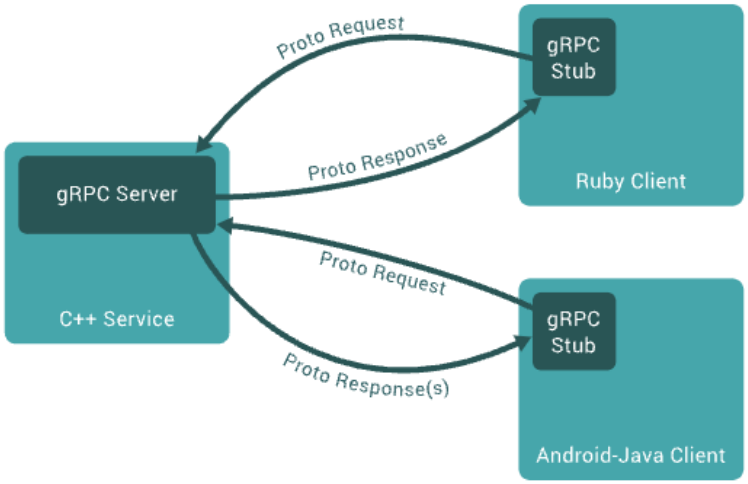
1. Caché de datos: La naturaleza flexible de GraphQL puede dificultar el uso de caché de datos pues debido a que la forma común en que GraphQL se utiliza sobre HTTP (Un POST en un único

endpoint), la caché a nivel de red se hace difícil. Una forma de resolverlo es utilizar Persisted Queries. (*Rest - Are There Any Disadvantages to GraphQL?*, n.d.). De todas formas, la mayoría de las librerías construidas sobre GraphQL ofrecen un mecanismo de caché eficiente. (*GraphQL Advantages and Disadvantages - Javatpoint*, n.d.)

2. Aumento en la carga del servidor: Hay que tener cuidado con las consultas complejas, como las anidadas ya que pueden conducir a consultas circulares y pueden bloquear el servidor con cargas adicionales. (*GraphQL Advantages and Disadvantages - Javatpoint*, n.d.)
3. La implementación de la descarga/carga de archivos es complicada: La conversión a cadena puede no ser la mejor opción para archivos grandes. (*Rest - Are There Any Disadvantages to GraphQL?*, n.d.)

gRPC

Al hablar de gRPC, este es un framework de comunicación remota desarrollado por Google y lanzado en 2016, que utiliza el Protocolo de Buffers (protobuf) como lenguaje de definición de interfaz (IDL) para escribir archivos de tipo esquema, .proto, que describe el servicio API. Especifica los métodos disponibles que pueden ser llamados y sus parámetros y tipos de retorno. Se basa en el paradigma de llamada a procedimientos remotos (RPC), donde los clientes pueden llamar directamente a un método de una aplicación servidor en una máquina diferente como si fuera un objeto local, lo que facilita la creación de aplicaciones y servicios distribuidos. En el lado del cliente, el cliente tiene un "stub" (denominado cliente en algunos lenguajes) que proporciona los mismos métodos que el servidor. (*GraphQL Vs gRPC | GraphQL Tutorial*, n.d.)



Por otro lado, gRPC se utiliza en arquitecturas de microservicios y comunicación entre sistemas distribuidos. Permite una comunicación eficiente y de alto rendimiento entre los diferentes servicios de

una aplicación. Al usar protobuf como lenguaje de interfaz, gRPC ofrece una serialización compacta y rápida de datos, lo que resulta en un menor uso de ancho de banda y menor latencia en comparación con otras tecnologías de comunicación remota.

Ventajas de gRPC:

1. Eficiente y rápido: Utiliza protobuf para la serialización de datos, lo que resulta en una comunicación eficiente con menor uso de ancho de banda y menor latencia, incluye también soporte nativo para la generación de código, entonces se puede generar código de servidor y cliente a partir del .proto sin necesidad de herramientas de terceros.
2. Multiplataforma: gRPC admite múltiples lenguajes de programación, lo que facilita la interoperabilidad entre diferentes sistemas y servicios.
3. Comunicación bidireccional y streaming: gRPC admite la comunicación bidireccional y la transmisión de datos en tiempo real, lo que es útil en casos de uso como chat en línea o transmisión de eventos. Lo que es clave de esta ventaja es que soporta server streaming, client streaming y bidirectional streaming. (*GraphQL Vs gRPC | GraphQL Tutorial*, n.d.)
    - *Server streaming*: el cliente hace una petición, y el servidor responde con un flujo de mensajes.
    - *Client streaming*: el cliente envía un flujo de mensajes al servidor, que responde después de que el cliente haya terminado el streaming
    - *Bidirectional streaming*: tanto el cliente como el servidor envían transmisiones independientes de mensajes entre sí.
    - *Unary interactions*: el cliente envía una petición, y el servidor envía una respuesta.

Desventajas de gRPC:

1. Acoplamiento más fuerte: Al utilizar gRPC, los clientes y servidores están acoplados mediante la definición del contrato en protobuf, lo que puede dificultar la evolución independiente de los servicios.
2. Un inconveniente es que Protobuf (búferes de protocolo), parte esencial de gRPC, solo admite código generado en 11 lenguajes: C#/.NET, C++, Dart, Go, Java, Kotlin, Node, Objective-C, PHP, Python y Ruby.
3. Dificultad en la depuración: La serialización binaria de protobuf puede dificultar la depuración de las comunicaciones gRPC en comparación con formatos de texto legibles por humanos, leer e inspeccionar los datos requiere pasos extras y herramientas

Demo: https://github.com/Cglassm/gRPC_demo

Conclusiones

Cuando se habla de gRPC y GraphQL en cuanto a obtención de datos, el segundo es más preciso debido a las consultas realizadas. Sin embargo, en cuanto a performance gRPC es mas rapido gracias al archivo protobuf y HTTP/2 pues los datos de la carga útil se serializan en formato binario, lo que reduce su tamaño y los hace más eficientes que los formatos basados en texto JSON o XML.

Si dispone de una API externa que a menudo requiere que los clientes externos realicen varias llamadas para obtener la información que necesitan, GraphQL puede ayudar a hacer que la API sea mucho más eficaz y sencilla. (Newman, 2021, 135).

Con gRPC, por su lado, se puede crear fácilmente un servidor gRPC en Java con clientes en Go, Python o Ruby. Las últimas API de Google tendrán versiones gRPC de sus interfaces, lo que permitirá incorporar fácilmente funciones de Google a tus aplicaciones. (*Introduction to gRPC*, 2023).

En caso de crear un chat, implementarlo con gRPC es una buena opción debido a la posibilidad de la comunicación bidireccional.

Para finalizar, en cuanto a soporte de comunidad, la de GraphQL es más amplia mientras que la de gRPC es más limitada. Ninguna tecnología es una solución única y también es posible combinar ambas para construir una solución mejor. Sin embargo, en el caso de optar por la comunicación *server-to-server,* donde es necesario más tipos de streaming, gRPC es mejor. Por otro lado, en la comunicación *client-server*, donde se quiere poder obtener todos los datos que se necesita en un solo viaje de ida y vuelta, tener flexibilidad en los datos que se obtienen para diferentes vistas y disponer de un potente almacenamiento en caché, GraphQL es claramente mejor.

Referencias

GraphQL Advantages and Disadvantages - javatpoint. (n.d.). Javatpoint. Retrieved June 25, 2023, from https://www.javatpoint.com/graphql-advantages-and-disadvantages

GraphQL vs gRPC | GraphQL Tutorial. (n.d.). Hasura. Retrieved June 25, 2023, from https://hasura.io/learn/graphql/intro-graphql/graphql-vs-grpc/

GraphQL vs REST | GraphQL Tutorial. (n.d.). Hasura. Retrieved June 25, 2023, from https://hasura.io/learn/graphql/intro-graphql/graphql-vs-rest/

Introduction to gRPC. (2023, February 16). gRPC. Retrieved June 25, 2023, from https://grpc.io/docs/what-is-grpc/introduction/

Newman, S. (2021). *Building Microservices Designing Fine-Grained Systems* (2nd ed.). O'Reilly.

Papa, G. (2022, December 26). *Real time apps with Flutter and gRPC*. Somnio Software. Retrieved June 27, 2023, from https://somniosoftware.com/post/how-to-build-real-time-apps-using-flutter-and-grpc

rest - Are there any disadvantages to GraphQL? (n.d.). Stack Overflow. Retrieved June 25, 2023, from https://stackoverflow.com/a/45538501

Sands, L., Donovan, R., & Williams, C. (2022, November 28). *When to use gRPC vs GraphQL*. Stack Overflow Blog. Retrieved June 25, 2023, from https://stackoverflow.blog/2022/11/28/when-to-use-grpc-vs-graphql/

Schemas and Types. (n.d.). GraphQL. Retrieved June 25, 2023, from https://graphql.org/learn/schema/

Schemas and Types. (n.d.). GraphQL. Retrieved June 25, 2023, from https://graphql.org/learn/schema/

What is GraphQL? | GraphQL Tutorial. (n.d.). Hasura. Retrieved June 25, 2023, from https://hasura.io/learn/graphql/intro-graphql/what-is-graphql/