GraphQL versus REST - Um comparativo entre as duas tecnologias

Átila Camurça Alves[1]

[1]Instituto Federal do Ceará (IFCE)

1. Introdução

A evolução dos computadores fez com que mais formas diferentes de acesso a informação sejam possíveis. Hoje em dia é possível, por exemplo, verificar sua conta de *e-mail* de seu computador pessoal, ou de seu *notebook*, de seu aparelho celular ou ainda de seu *smartwatch*. Cada um desses aparelhos com diferentes tamanhos físicos e diferentes poderes de processamento. Nesse cenário, voltamos um pouco da ideia de *Mainframes* para acesso remoto de arquivos ou serviços, mas agora com novas estruturas e arquiteturas, que ficou conhecido como Nuvem.

Para atender a estes diversos tipos de dispositivos citados acima que acessam aos mesmos recursos surge o conceito de Serviços. O que se propõe é tornar o servidor e o cliente independentes — no sentido de existir um *back-end* e diferentes *front-ends*, para que seja possível ter acesso aos mesmos recursos do servidor de vários clientes diferentes.

2. Definição

Em 2000, surge o conceito do REST (do inglês *Representational State Transfer*) criado por Roy Thomas Fielding em sua dissertação de Doutorado. Uma dos pontos-chave deste conceito é desacoplar a interface de usuário do armazenamento de dados, o que melhoraria a portabilidade da interface de usuário para múltiplas plataformas [Fielding 2000]. Esse padrão se popularizou graças a adoção por empresas como o *Twitter* em 2006, que fornecedia uma API (do inglês *Application Programming Interface*) para que terceiros pudessem fazer algum tipo de integração, como por exemplo *login* com a conta do *Twitter* em seu aplicativo ao invés de criar um sistema de *login* próprio.

Em 2016, o *Facebook* publica em formato *Open Source* o GraphQL, definido como uma *query language* (linguagem de consulta) para *API*'s. Permite que o cliente obtenha os dados de forma declarativa e hierarquica [Facebook 2016]. É um formato que adapta-se a qualquer banco de dados ou mecanismo de armazenamento, ou seja, é possível, por exemplo, usá-lo para busca de arquivos em um sistema de arquivos remoto. Possui 3 tipos de operações básicas: *Query* para obtenção de dados; *Mutation* para alteração de dados; e *Subscriptions* para notificação em tempo real.

O artigo "GraphQL vs REST: Overview" descreve com mais detalhes outras diferenças e características [Sturgeon 2017].

3. Estilo Arquitetônico e Arquitetura

3.1. REST

Seu estilo arquitetônico se encaixa em uma arquitetura em camadas, como pode ser visto na Figura 1. Em sistemas REST há série de restrições arquiteturais, tais como:

- **Cliente-Servidor**: separar a interface de usuário da lógica da aplicação.
- ***Stateless***: a requisição contém informações suficientes sobre o cliente para que o servidor entenda a requisição.
- ***Cacheable***: requisições devem ser passíveis de serem colocadas em *cache*.
- **Interface Uniforme**: identificação de recursos é feita através de URI e sua implementação é feita de forma abstrata da definição da interface.
- **Sistema em Camadas**: um cliente não deverá distinguir se está conectado ao servidor final ou a um intermediário.

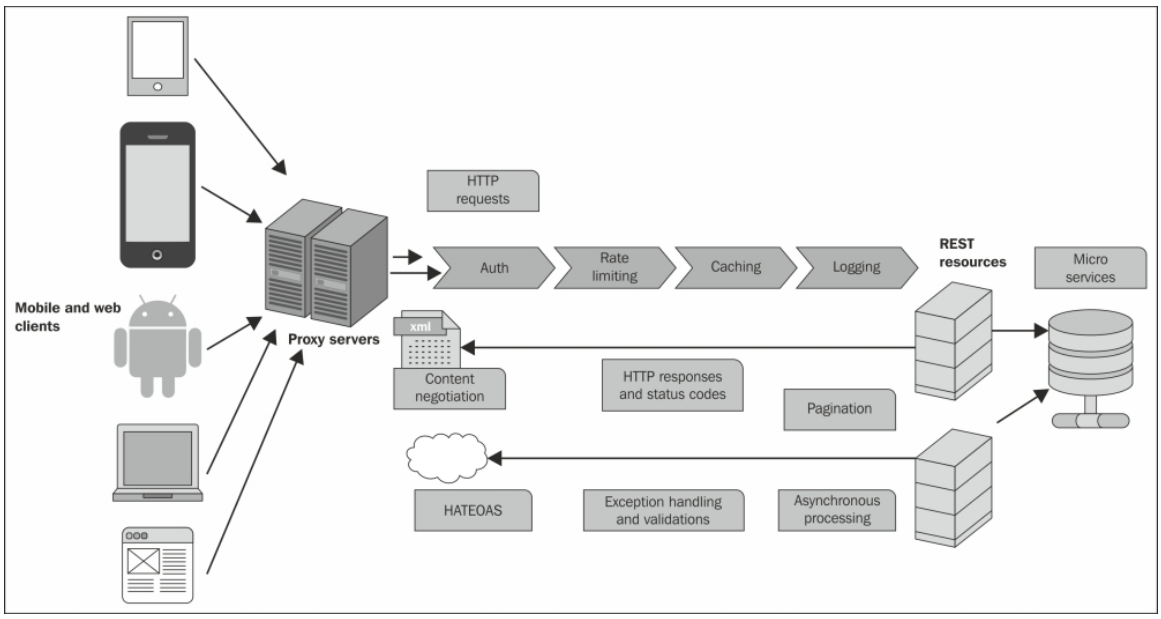

Figura 1. Arquitetura REST

3.2. GraphQL

Seu estilo arquitetônico é híbrido entre arquitetura em camadas e baseada em eventos, esta última utilizada para notificações em tempo real [Howtographql 2017]. Em sistemas GraphQL podem haver diversos tipos de arquiteturas diferentes, entre elas:

- Servidor GraphQL conectado a um banco de dados, como visto na Figura 2.
- Servidor GraphQL atuando como camada intermediária entre sistemas legados ou ainda de terceiros integrando-os através de uma única API GraphQL, como visto na Figura 3.
- Uma abordagem híbrida com a junção das 2 abordagem supracitadas, como visto na Figura 4.

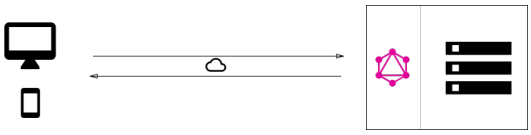

Figura 2. GraphQL com banco de dados

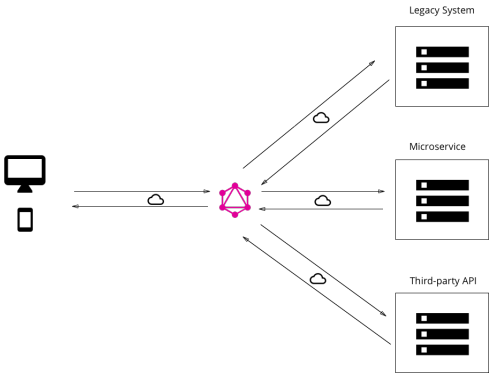

Figura 3. GraphQL com sistemas de terceiros ou legados

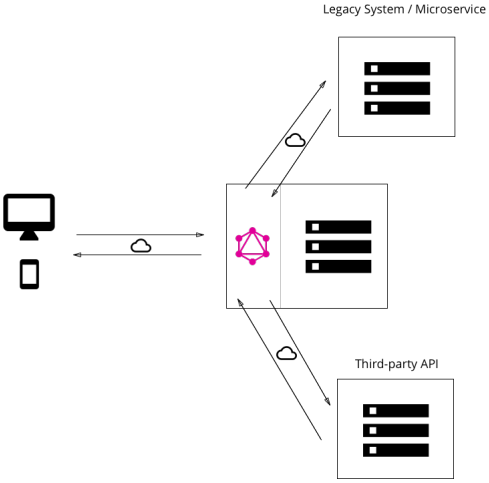

Figura 4. GraphQL híbrido

4. Comunicação

4.1. REST

A comunicação é feita através do protocolo HTTP, em que é recomendado o uso explícito dos métodos do HTTP para operações: criar, ler, atualizar e deletar [Rodriguez 2008], como é mostrado a seguir:

- Para criar um recurso no servidor, utilize o método `POST`;
- Para resgatar um recurso, use `GET`;
- Para modificar um recurso, use `PUT`;
- Para deletar um recurso, use `DELETE`.

4.2. GraphQL

A comunicação pode ser feita por qualquer protocolo, basta que para isso seja implementado no servidor. A única restrição é que haja somente um *endpoint* para a conversa entre cliente e servidor.

5. Nomeação

Ambas tecnologias usam sistema de nomeação estruturada, mais especificamente o DNS (do inglês *Domain Name System*). Entretanto o REST utiliza *n-endpoint*'s, em que cada um deles retornam um recurso diferente; e no GraphQL existe apenas um *endpoint*, em que é passado a *query* com a definição dos recursos a serem retornados.

5.1. Exemplo usando REST

Suponhamos que a URL de acesso seja `api.company.com`, e o recurso **produto**. Teremos:

- [`GET`] `api.company.com/produtos` - Obtem uma lista de produtos
- [`GET`] `api.company.com/produto/:id` - Obtem um produto específico
- [`POST`] `api.company.com/produto` - Salva um novo produto
- [`PUT`] `api.company.com/produto/:id` - Atualiza um produto existente
- [`DELETE`] `api.company.com/produto/:id` - Deleta um produto

5.2. Exemplo usando GraphQL

Suponha a mesma URL do exemplo anterior e o mesmo recurso. Para obter uma lista de produtos devemos enviar a *query* abaixo para o *endpoint* [`GET` ou `POST`] `api.company.com/graphql`, teremos:

```
query {
    produtos(first: 10) {
        edges {
            node {
                id
                descricao
                grupo_mercadoria {
                    descricao
                }
            }
        }
        pageInfo {
            hasNextPage
            hasPreviousPage
            startCursor
            endCursor
        }
    }
}
```

6. Dificuldades e Soluções

Criar uma API usando REST chega até a ser algo trivial hoje em dia. São muitas *frameworks* e exemplos em todas as linguagens de programação, como JavaScript, Java, PHP, Python, Ruby, etc [Marmelab 2017]. Isso se deve a ampla adoção do padrão ao longo dos anos.

Antes do GraphQL outras soluções semelhantes surgiram para otimizar a busca de recursos estendendo o padrão REST. Entretanto nenhuma delas chegou a se tornar um padrão ou mesmo ser adotada em larga escala. Somente com a chegada do GraphQL foi possível enxergar um novo padrão surgir e que fizesse sentido para ser implementado nos dias atuais, com o crescente uso de diversos dispositivos de acesso à *Internet*.

Mesmo sendo um padrão relativamente novo o GraphQL teve rápida adoção por empresas como o GitHub, que trocou sua API REST por GraphQL. Outro ponto importante é a ótima documentação criada para explicar seu funcionamento. Tenha em mente que o GraphQL é apenas a especificação da linguagem, para a execução é preciso ter um servidor e um cliente que conversem através do GraphQL ou *Graph Query Language*. Nesse sentido uma empresa que surgiu para atender essa necessidade foi a Apollo Data, que criou um conjunto de ferramentas para desenvolvimento de servidores e clientes GraphQL, até mesmo com a possibilidade de trabalhar em conjunto com serviços REST existentes [Apollodata 2017b].

E assim foi feito para este trabalho, foi usado o servidor REST `restify` [Restify 2017] e o `apollo-server` para integração com GraphQL [Apollodata 2017a]. Para a resolução

das *queries* e mapeamento com as colunas do banco de dados foi utilizada a ferramenta `join-monster`, que traduz GraphQL para SQL de forma eficiente [Stem 2017]. Já para o servidor REST foi utilizada a biblioteca `bookshelfjs`, que é uma ferramenta ORM (do inglês Object-Relational Mapping) que auxilia na busca de dados do banco para cada *endpoint* [Griesser 2017].

7. Considerações Finais

Apesar de, às vezes, ser considerada o substituto do REST, o GraphQL não deve ser utilizado dessa maneira apenas por ser algo novo e inovador. Uma de suas características é exatamente ser capaz de trabalhar em conjunto com tecnologias existentes e assim deve ser feito.

Tomemos o exemplo do GitHub que substituiu por completo sua API (ou v3) em REST (que por sinal ainda está em funcionamento) [GitHub 2017b] por uma API GraphQL (ou v4) [GitHub 2017a]. No caso deles fez muito sentido a mudança para apenas uma tecnologia, pois é o tipo de API que será usada por terceiros e que podem ter necessidade de recursos diferentes, por exemplo um aplicativo que mostre os repositórios favoritos ao redor do mundo com as pessoas envolvidas detalhadamente; e outro que apenas diz as 10 linguagens mais usadas nos projetos. É notável que os dois projetos irão utilizar informações distintas e que um é mais detalhado do que o outro.

Agora imagine uma API interna de uma empresa. Nesse caso sabemos de antemão quais serão os recursos necessários e podemos criá-los de forma otimizada. Em um segundo momento a empresa cria um aplicativo *mobile* e que deve consumir o mínimo de banda possível, isso indicaria a necessidade de criação de um *endpoint* GraphQL para atender essa demanda sem ser necessário refazer a API REST existente. A ideia é utilizar GraphQL em situações em que se busca utilização mínima ou dinâmica de recursos.

Referências

Apollodata (2017a). GraphQL server for Restify, Express, Connect, Hapi and Koa. `https://github.com/apollographql/apollo-server`. [Online; accessed 23-September-2017].

Apollodata (2017b). Tools Products for GraphQL. `https://www.apollodata.com/`. [Online; accessed 23-September-2017].

Facebook (2016). GraphQL - A query language for your API. `http://graphql.org/`. [Online; accessed 23-September-2017].

Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. `https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm`. [Online; accessed 23-September-2017].

GitHub (2017a). GitHub GraphQL API v4. `https://developer.github.com/v4/`. [Online; accessed 23-September-2017].

GitHub (2017b). GitHub REST API v3. `https://developer.github.com/v3/`. [Online; accessed 23-September-2017].

Griesser, T. (2017). A simple Node.js ORM for PostgreSQL, MySQL and SQLite3 built on top of Knex.js. `http://bookshelfjs.org/`. [Online; accessed 23-September-2017].

Howtographql (2017). Big Picture (Architecture). https://www.howtographql.com/basics/3-big-picture/. [Online; accessed 23-September-2017].

Marmelab (2017). Awesome REST - Servers. https://github.com/marmelab/awesome-rest#servers. [Online; accessed 23-September-2017].

Restify (2017). The future of Node.js REST development. http://restify.com/. [Online; accessed 23-September-2017].

Rodriguez, A. (2008). RESTful Web services: The basics. https://www.ibm.com/developerworks/library/ws-restful/index.html. [Online; accessed 23-September-2017].

Stem (2017). Query Planning and Batch Data Fetching between GraphQL and SQL. http://join-monster.readthedocs.io/en/latest/. [Online; accessed 23-September-2017].

Sturgeon, P. (2017). GraphQL vs REST: Overview. https://philsturgeon.uk/api/2017/01/24/graphql-vs-rest-overview/. [Online; accessed 23-September-2017].