GraphQL as an alternative approach to REST
About me

Luis Weir
CTO at Capgemini UK Oracle
Ace Director & Groundbreaker Ambassador

luis.weir@capgemini.com
uk.linkedin.com/in/lweir
http://www.soa4u.co.uk

Want the slides? I’ll share via:

@luisw19

Latest articles:

- The 7 Deadly Sins of API Design
- Setting the vision, strategy and direction — the CTO’s role
- How can you design, deploy and manage your APIs?
- The Spotify’s Engineering Culture. My interpretation and summary
- A comparison of API Gateways communication styles
- Is BPM Dead, Long Live Microservices?
- Five Minutes with Luis Weir
- 2nd vs 3rd Generation API Platforms - A Comprehensive Comparison
- Podcast: Are Microservices and APIs Becoming SOA 2.0?
- 3rd-Generation API Management: From Proxies to Micro-Gateways
- Oracle API Platform Cloud Service Overview
GraphQL as an alternative approach to REST

01. GraphQL – context & key concepts
02. Demos
03. GraphQL vs REST PoV
04. Conclusions
Application Programming Interfaces (APIs) are doors to information and functionality.
But some doors can be unfit for purpose...

source: https://imgur.com/a/J3ttg
Why GraphQL?

Source: https://dev-blog.apollodata.com/graphql-vs-rest-5d425123e34b by Sashko Stubailo
GraphQL - Background

- Created by Facebook in 2012 to get around a common constraints in the REST approach when fetching data
- Publicly released in 2015
- GraphQL Schema Definition Language (SDL) added to spec in Feb’18

https://GraphQL.org

Latest release: http://facebook.github.io/graphql
Latest draft: http://facebook.github.io/graphql/draft/
GraphQL – What is it **NOT**?

in spite of its name, it has nothing to do with Graphs DBs

a query language for a databases

necessarily a replacement for REST. Both can work together

A silver Bullet

Roy from the IT Crowed ➔
GraphQL – What is it then?

A consumer oriented **query language**, a strongly typed **schema language** and a **runtime** to implement **GraphQL services**.

**Define Schema**

```typescript
type Country {
  id: ID!
  name: String!
  code: String!
}

type query {
  countries: [Country]
}
```

**GraphQL Service**

**Quickly write and run queries**

```typescript
{
  getCountries(name:"great")
  {
    name
  }
}
```

**GraphQL Client**

**Get exactly what you asked for**

```typescript
{
  "data": {
    "countries": [
    {
      "name": "United Kingdom"
    }
    ]
  }
}
```
Who is using it?

Lots of organisations are embracing GraphQL: http://graphql.org/users
Increasing rapidly in Popularity

https://trends.google.com/trends/explore?date=2016-01-01%202018-12-01&q=GraphQL,REST,OData
Let’s put it into perspective

https://trends.google.com/trends/explore?date=2004-01-10%202018-11-30&q=GraphQL,REST%20API,OData,WSDL
GraphQL – Key Concepts

There are 5 key characteristics of GraphQL that are important to understand:

1. **Hierarchical**
   Queries as hierarchies of data definitions, shaped just how data is expected to be retuned.

2. **View-centric**
   By design built to satisfy frontend application requirements.

3. **Strongly-typed**
   A GraphQL server defines a specific type system. Queries are executed within this context.

4. **Introspective**
   The type system itself is queryable. Tools are built around this capability.

5. **Version-Free**
   GraphQL takes a strong opinion on avoiding versioning by providing the tools for the continuous evolution.
GraphQL – Anatomy

GraphQL Schema

GraphQL Definition Language

GraphQL Schema

Define Types

(Data) Types
(Object, Input, Scalar, Enum, Interface, Union)

Queries
(operation type)

have

Mutations
(operation type)

have

Subscriptions
(operation type)

have

Resolve

f(x), f(x), f(x)

Execute

GraphQL Server Implementation
Effectively the blueprint of a GraphQL API, it defines the Types, Queries and Mutations supported in a GraphQL service.
GraphQL – Anatomy

GraphQL Schema
Definition Language

GraphQL Schema

Define Types

2. Data) **Types**
   (Object, Input, Scalar, Enum, Interface, Union)

   have

   **Queries**
   (operation type)

   have

   **Mutations**
   (operation type)

   have

   **Subscriptions**
   (operation type)

   have

   **Resolvers**
   \( f(x), f(x), f(x) \)

   Execute

Object, Input, Scalar, Enumeration, Interfaces and Unions are all types to define data structures, which are used in Operation types.

Implementation

@luisw19
GraphQL – Anatomy

Entry point for operations that fetch data (read / search). Note that a single Query type can define multiple query operations.
GraphQL – Key Concepts

GraphQL Schema
Definition Language

Entry point for operations that create/update data via a GraphQL service. Note that a single Mutation type can define multiple mutation operations.
GraphQL – Anatomy

**GraphQL Schema Definition Language**

**GraphQL Schema**

Define Types

(Data) **Types**
(Object, Input, Scalar, Enum, Interface, Union)

have

**Queries**
(operation type)

have

**Mutations**
(operation type)

have

**Subscriptions**
(operation type)

have

**Resolvers**

*f(x), f(x), f(x)*

*new* Pub/sub system for near-realtime updates. Unlike queries or mutations, it can deliver more than one result via push.
GraphQL – Anatomy

GraphQL Schema

GraphQL Schema Definition Language

Define Types

(Data) Types
(Object, Input, Scalar, Enum, Interface, Union)

Queries
(operation type)

Mutations
(operation type)

Subscriptions
(operation type)

Functions that define how each field, within a GraphQL Query or Operation is to be acted upon.

Resolvers
f(x), f(x), f(x)

GraphQL
GraphQL Schema Cheat Sheet

https://github.com/sogko/graphql-schema-language-cheat-sheet
Simple GraphQL Query Demo (I) - Mock

https://github.com/luisw19/graphql-samples/tree/master/graphql-countries-part1

GraphQL Client

Browser

Query Operation {JSON}
[HTTP/POST]

{JSON}

http://.../graphiql

GraphQL Endpoint

GraphQL Schema

Graphiql

GraphQL Server:
Apollo/Express

Apollo Express: https://github.com/apollographql/apollo-server
Simple GraphQL Query Demo (II) – REST Backend

https://github.com/luisw19/graphql-samples/tree/master/graphql-countries-part2

GraphQL Client

Query Operation

[HTTP/POST]

{ query
  data }

Browser

GraphQL Endpoint

Graphiql

GraphQL Schema

HTTP/GET

https://restcountries.eu/rest/v2/{resource}

REST COUNTRIES

Query Operation

{JSON}

{JSON}

https://github.com/luisw19/graphql-samples/tree/master/graphql-countries-part2
Simple GraphQL Query Demo (III) - Mutation

https://github.com/luisw19/graphql-samples/tree/master/graphql-countries-part3

GraphQL Client

```
{
  query
    data
}
```

GraphQL Service

- GraphQL Endpoint
- GraphQL Schema
- Graphiql

REST COUNTRIES

- [HTTP/GET] https://restcountries.eu/rest/v2/{resource}

Docker Container

Graphiql

Browser

Query Operation [HTTP/POST] {JSON}

Request BIN

https://github.com/luisw19/graphql-
samples/tree/master/graphql-
countries-part3
Simple GraphQL Query Demo (IV) – API Composition

Not merged yet... (soon)

GraphQL Client

Browser

Query Operation

{HTTP/POST}

{JSON}

{query}

{data}

GraphQL Endpoint

{JSON}

GraphQL Service

GraphQL Server:
Apollo/Express

REST COUNTRIES

[HTTP/GET]
https://restcountries.eu/rest/v2/{resource}

{JSON}

Google

[HTTP/POST]
https://www.google.co.uk/search?q={search}

{JSON}

02
GraphQL vs REST

**Developer Experience** (design and consume APIs)

- **Design**
  - 

- **Build**
  - 

- **Try**
  - 

(+++) Usage: Best usage experience for developers (Graphiql is brilliant!)
(~) API-first design: Tooling evolving (build a service to mock).

(~) Usage: depends on the quality of API definition and documentation
(+ ) API-first design: good tools available (e.g. Apiary, Swagger Hub).

**API Gateway** (API routing, security, policies)

- **API Gateway**

(-) Existing Gateways have rich support for REST, not yet GraphQL - but could be used. Alternative is to use a GraphQL Service as API Gateway.

(+++) API Gateways take away from REST endpoints common tasks (e.g. OAuth, API keys, throttling, security).

**API Composition** (query data from multiple sources)

- **API**

(+++) Perfectly suited for API composition. Each field can be fetch (in parallel) from any source in a single query.

(-) The nature of REST makes it difficult to model resources that combine data structures from multiple sources. HATEOAS results in chattiness.
GraphQL vs REST

<table>
<thead>
<tr>
<th>Authentication / Authorization</th>
<th>Caching</th>
<th>Versioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client → Authorisation Server</td>
<td>Client → Network Caching → App Cache → Resource → Back End</td>
<td>(+) Best practices are clear. Versioning should be avoided and tools are provided (e.g. deprecation of fields) for continuous evolution.</td>
</tr>
<tr>
<td>(-) Standards like OAuth, OpenID can be used however as all ops can be accessed by single URI, custom authorisation is typically required.</td>
<td>(-) Network: unsuitable as there is a common URL for all operations. (+) Service: possible based on Object Type (even fields) and in mem cache like REDIS.</td>
<td>(-) Best practice less clear, in practice URI based versioning very popular although not encouraged.</td>
</tr>
<tr>
<td>(+) Major standards (OAuth 2, OpenId) supported by API Gateways and frameworks.</td>
<td>(+) Network: Caching is easy as each resource is a unique URI. Common tools can be used (e.g. CDNs). (+) Service: It’s equally possible at service Level.</td>
<td>(+++) Best practices are clear. Versioning should be avoided and tools are provided (e.g. deprecation of fields) for continuous evolution.</td>
</tr>
</tbody>
</table>
# GraphQL vs REST (completely subjective!)

<table>
<thead>
<tr>
<th></th>
<th>GraphQL</th>
<th>REST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>++++ (7)</td>
<td>++++ (8)</td>
</tr>
<tr>
<td></td>
<td>~ (1)</td>
<td>~~ (2)</td>
</tr>
<tr>
<td></td>
<td>---- (4)</td>
<td>--- (3)</td>
</tr>
</tbody>
</table>

But it will only improve!
Conclusions

Still early days but GraphQL has huge potential
GraphQL takes away many of the headaches of dealing with REST from a client side -specially around complex queries (against multiple sources). However tooling specially around API Design and API Gateways is still evolving. So bear this in mind when considering GraphQL.

GraphQL and REST can work nicely together
There are thousands of REST APIs (external and internal) and REST still is a viable and popular option. Instead of boiling the ocean, as Roy said, GraphQL is not necessarily a replacement for REST. As shown in this presentation both can be complementary and work together nicely.

There is no silver bullets –do your own research
There is tons of information available in the GraphQL Communities page. Explore it, learn about it and adopt it based on your own criteria and requirements. And hope this presentation helps in the process!
Resources

- GraphQL as an alternative approach to Rest recording at Devoxx’18 London
  https://www.youtube.com/watch?v=hJOOdCPlXbU

- Github repository with the GraphQL tutorials
  https://github.com/luisw19/graphql-samples

- Related articles:
  - GraphQL with Oracle Database and node-oracledb by Christopher Jones
    https://blogs.oracle.com/opal/demo%3a-graphql-with-node-oracledb
  - GraphQL+OracleDB by Steven B
    https://github.com/cloudsolutionhubs/oracledb-graphql-demo
With more than 190,000 people, Capgemini is present in over 40 countries and celebrates its 50th Anniversary year in 2018. A global leader in consulting, technology and outsourcing services, the Group reported 2016 global revenues of EUR 12.5 billion. Together with its clients, Capgemini creates and delivers business, technology and digital solutions that fit their needs, enabling them to achieve innovation and competitiveness. A deeply multicultural organization, Capgemini has developed its own way of working, the Collaborative Business Experience™, and draws on Rightshore®, its worldwide delivery model.

Learn more about us at
www.capgemini.com