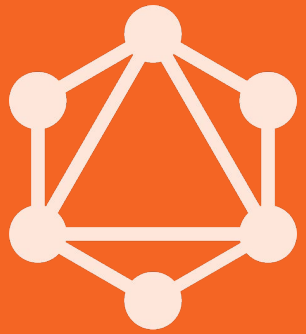


GraphQL

a data language and new paradigm for building APIs

A primer for software { architects, developers }

GenevaJUG · Tech talks · 29.10.2019 · bit.ly/gql-primer

Outline of the talk

- ✣ What is GraphQL?
- ❇ A brief history
- ♖ Architect's Hat
- ❀ Risk & Opportunities
- ♕ Best practices
- ❊ Further reading & notes

About the author

My name is Olivier Lange. I build apps & *ad hoc* software, hand-in-hand with software teams, providing support with app design, data- and process modelling. Digital archives · Data migration · Graphs & visualizations.

What is GraphQL?

Describe your data

```
type Project {
  name: String
  tagline: String
  contributors: [User]
}
```

Ask for what you want

```
{
  project(name: "GraphQL") {
    tagline
  }
}
```

Get predictable results

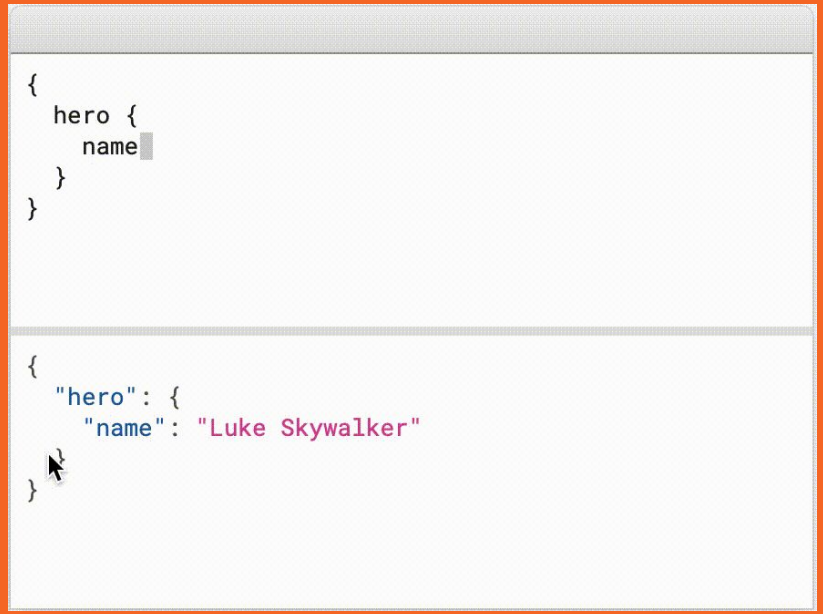
```
{
  "project": {
    "tagline": "A query language for APIs"
  }
}
```

What is GraphQL?

A query language for APIs and **a runtime** for fulfilling those queries

The **shape of a query** mirrors the **shape of the data** it returns.

« Send a GraphQL query to your API and get exactly what you need, nothing more and nothing less. »

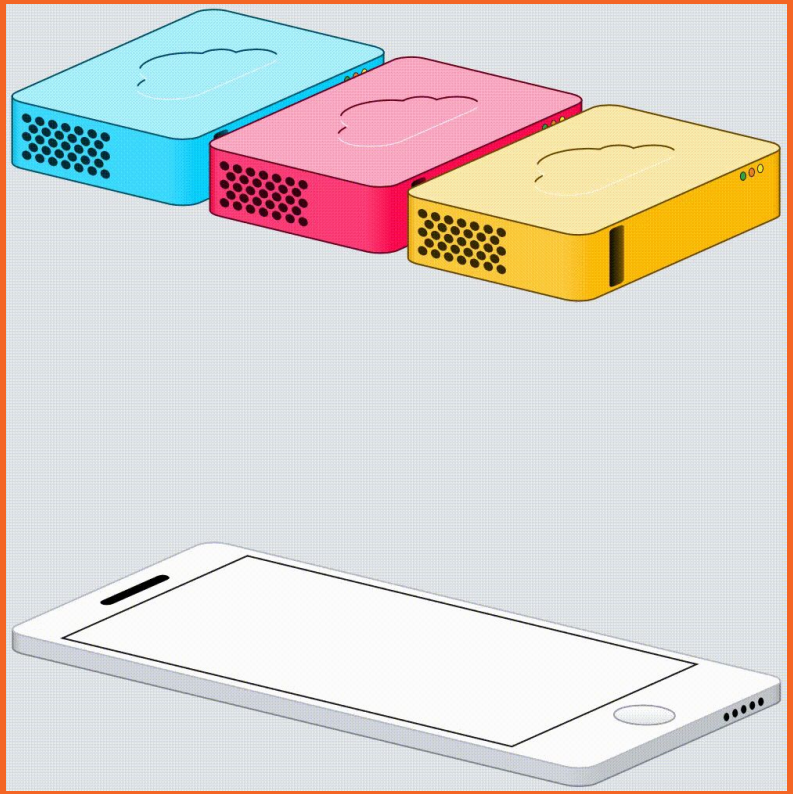


What is GraphQL?

Get many resources in a single request

« GraphQL queries access not just the properties of one resource but also smoothly follow references between them.

While typical REST APIs require loading from multiple URLs, GraphQL APIs get all the data your app needs in a single request. »



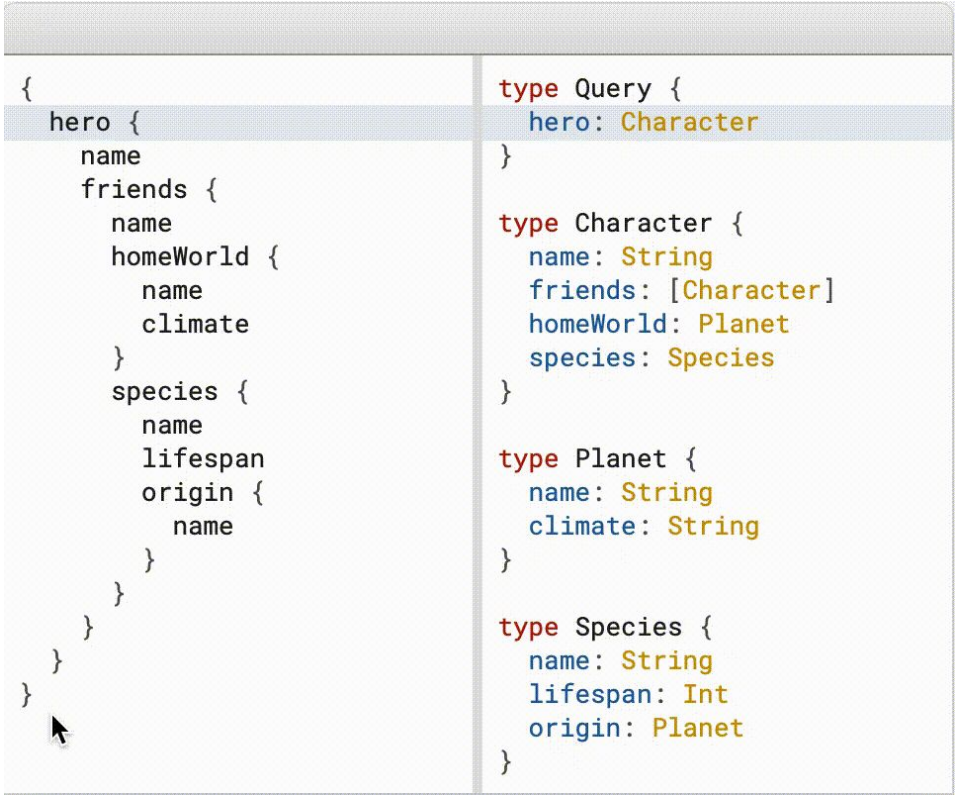
What is GraphQL?

GraphQL APIs are organized in terms of types and fields — not endpoints

The **schema** defines an API's type system and all object relationships.

Allowed operations: **queries** and **mutations**.

Schema-defined types: **scalars**, **objects**, **enums**, **interfaces**, **unions**, and **input objects**.



```
{
  hero {
    name
    friends {
      name
      homeWorld {
        name
        climate
      }
      species {
        name
        lifespan
        origin {
          name
        }
      }
    }
  }
}
```

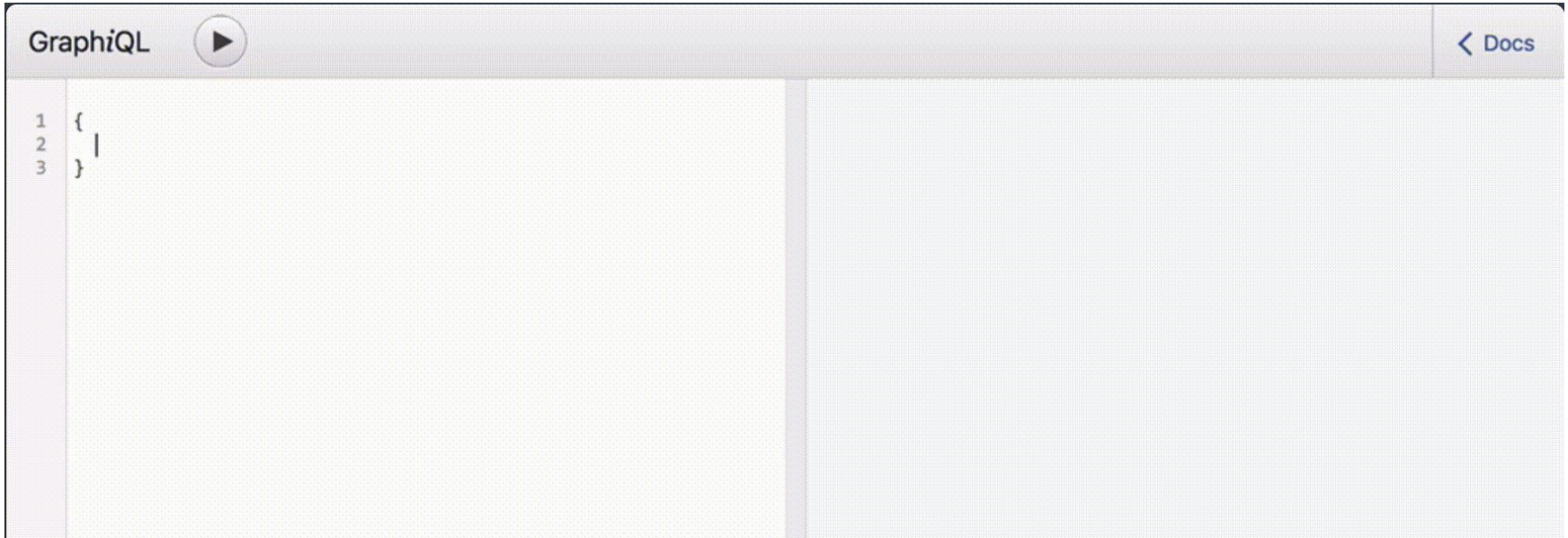
```
type Query {
  hero: Character
}

type Character {
  name: String
  friends: [Character]
  homeWorld: Planet
  species: Species
}

type Planet {
  name: String
  climate: String
}

type Species {
  name: String
  lifespan: Int
  origin: Planet
}
```

What is GraphQL?

Powerful query interface leveraging your type system

Introspection — a client can query the *schema* for details about the schema. Documentation is self-generated and always *up-to-date*.

Hierarchical — nested fields let you query for and receive only the data you specify in a single round trip.



What is GraphQL?

Evolve your API without versions

```
type Film {
  title: String
  episode: Int
  releaseDate: String



}
```

Add new fields and **types** to your GraphQL API without impacting *existing queries*.

Aging fields can be *deprecated* and hidden from tools.

By using a **single evolving version**, GraphQL APIs give apps continuous access to *new features* and encourage cleaner, more *maintainable* server code.

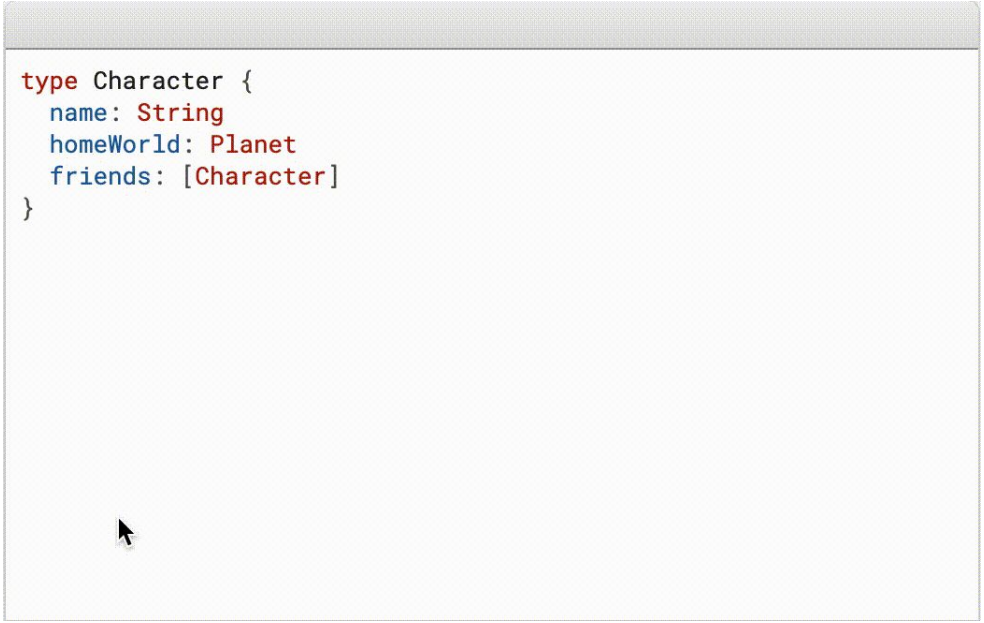
What is GraphQL?

Bring your own data and code

« *GraphQL creates a* ***uniform API across*** *your entire application without being limited by a specific* storage engine.

Write GraphQL APIs *that leverage* ***your existing data and code*** *with* GraphQL engines *available in many languages.*

You provide ***resolver functions*** *for each* field *in the type system, and GraphQL calls them with optimal concurrency.* »



What is GraphQL?

Not a storage model or database query language

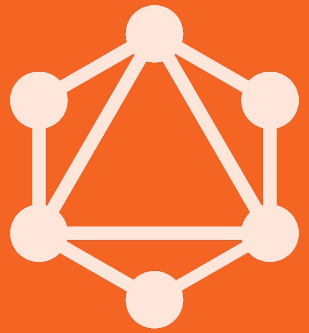
« The *graph* refers to *graph structures* defined in the *schema*, where nodes define objects and edges define relationships between objects.

The API traverses and returns application data based on the schema definitions, independent of how the data is stored. »

```
{
  human(id: 1002) {
    name
    appearsIn
    starships {
      name
    }
  }
}
```

```
{
  "data": {
    "human": {
      "name": "Han Solo
      "appearsIn": [
        "NEWHOPE",
        "EMPIRE",
        "JEDI"
      ],
      "starships": [
        {
          "name": "Mill
        },
        {
          "name": "Impe
        }
      ]
    }
  }
}
```

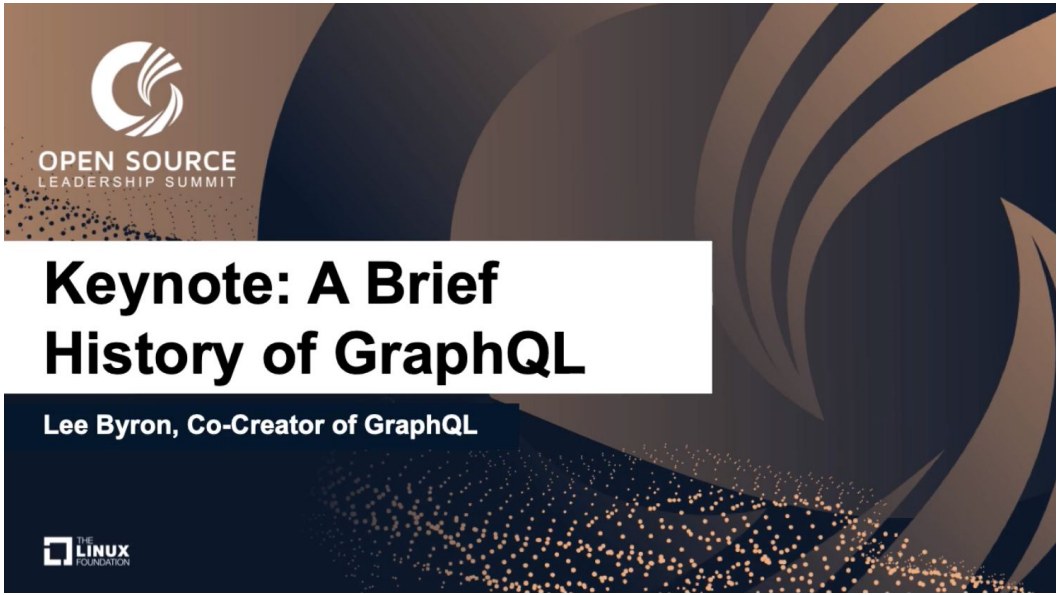
A brief history of GraphQL



Brief history

« **A Brief history of GraphQL** » by Lee Byron

YouTube · video 11mn. · 15.03.2019



Facebook Mobile app in 2012
Native iOS view backed by a RESTful API

- Slow on network — roundtrips
- Fragile client↔server relationship
- API docs out-of-date, tedious code & process

Lee Byron, Dan Schafer and Nick Schrock
First principles approach — GraphQL

- Fast on network — single network roundtrip
- Robust static types
- Empowering client evolution

Brief history

« **A Brief history of GraphQL** » by Lee Byron

YouTube · video 11mn. · 15.03.2019

2013 — React.js open-sourced

- Data Fetching for React Applications
- Lot of interest for Relay
- Needed release of GraphQL (internal)

2015 —Relay & GraphQL open-sourced

- GraphQL Specification
- GraphQL.js reference implementation
- 6 alternative implementations

2019 —Worldwide community

- Major conferences on every continent
- Worldwide chapters, not linked to FB
- GraphQL was developed into stable base

GraphQL Foundation

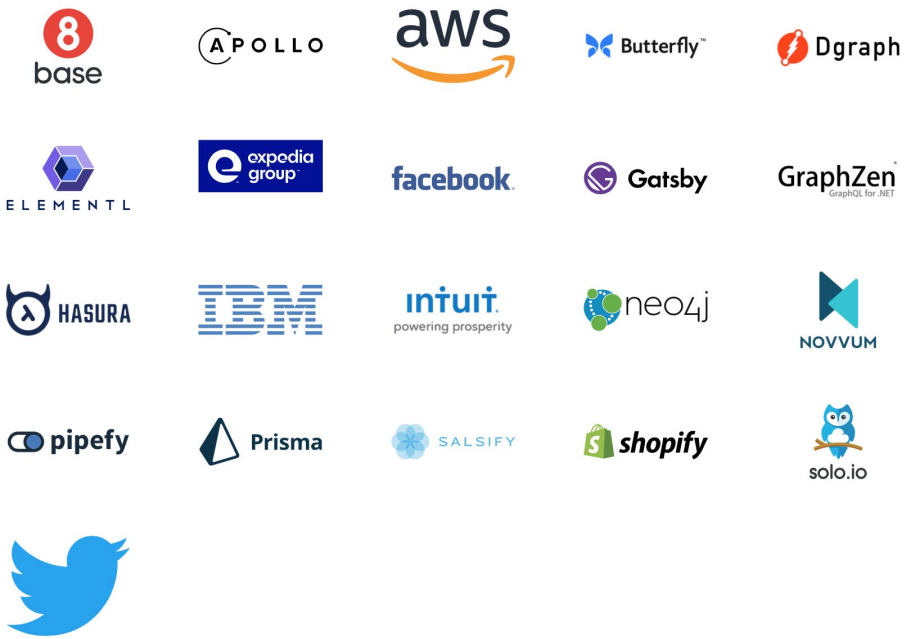
- Accelerate GraphQL standards
- Open collaboration and collaboration
- Help fund community initiatives

Brief history

GraphQL Foundation & non-profit organization

A neutral foundation founded by global technology and application development companies.

« The GraphQL Foundation encourages contributions, stewardship, and a shared investment from a broad group in vendor- neutral events, documentation, tools, and support for GraphQL. »



Background

Who's using GraphQL?

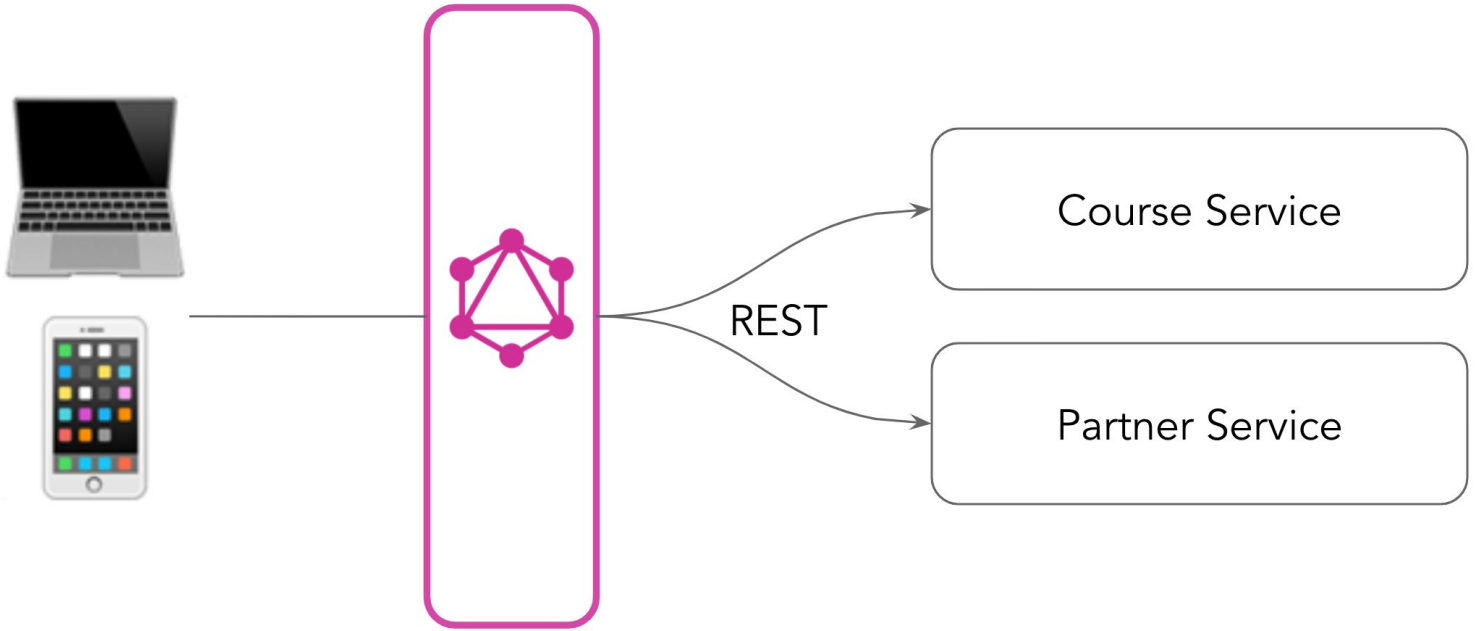
Seriously!

https://developer.github.com/v4/

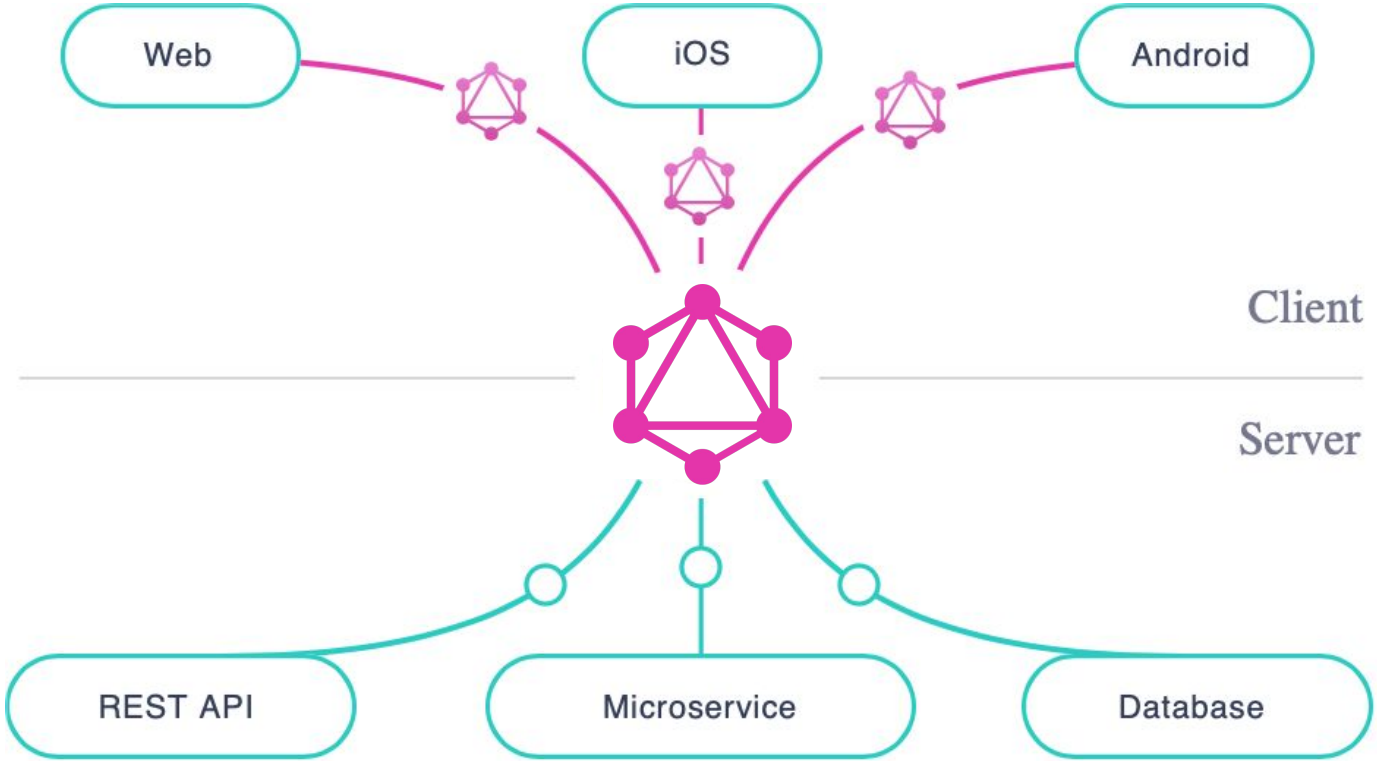
https://developer.github.com/v4/explorer/

Who's using GraphQL?

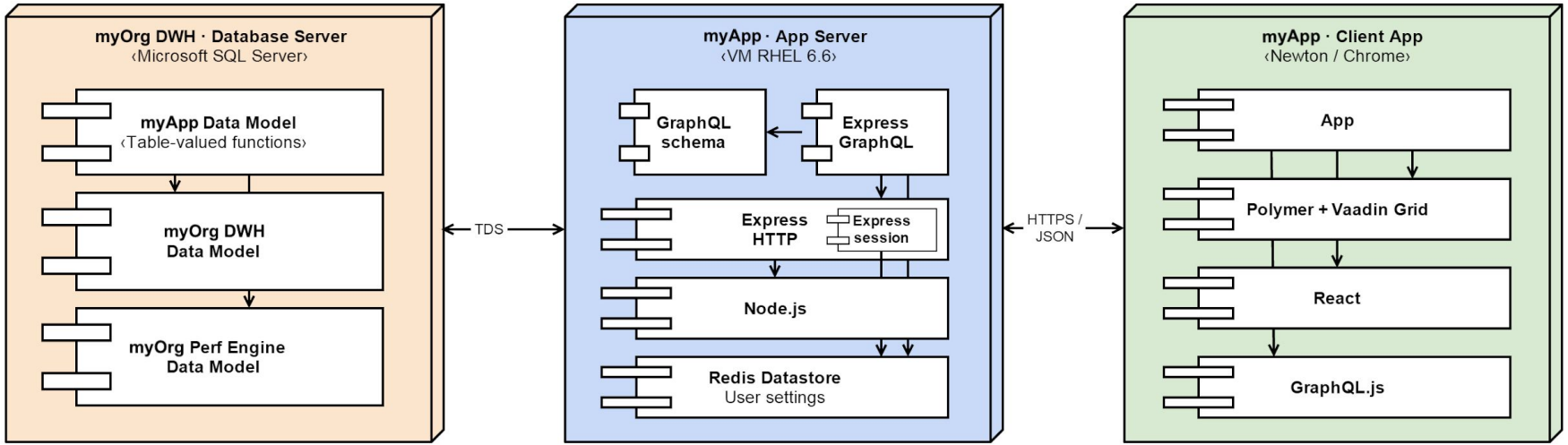
Put on your **Architect's Hat**



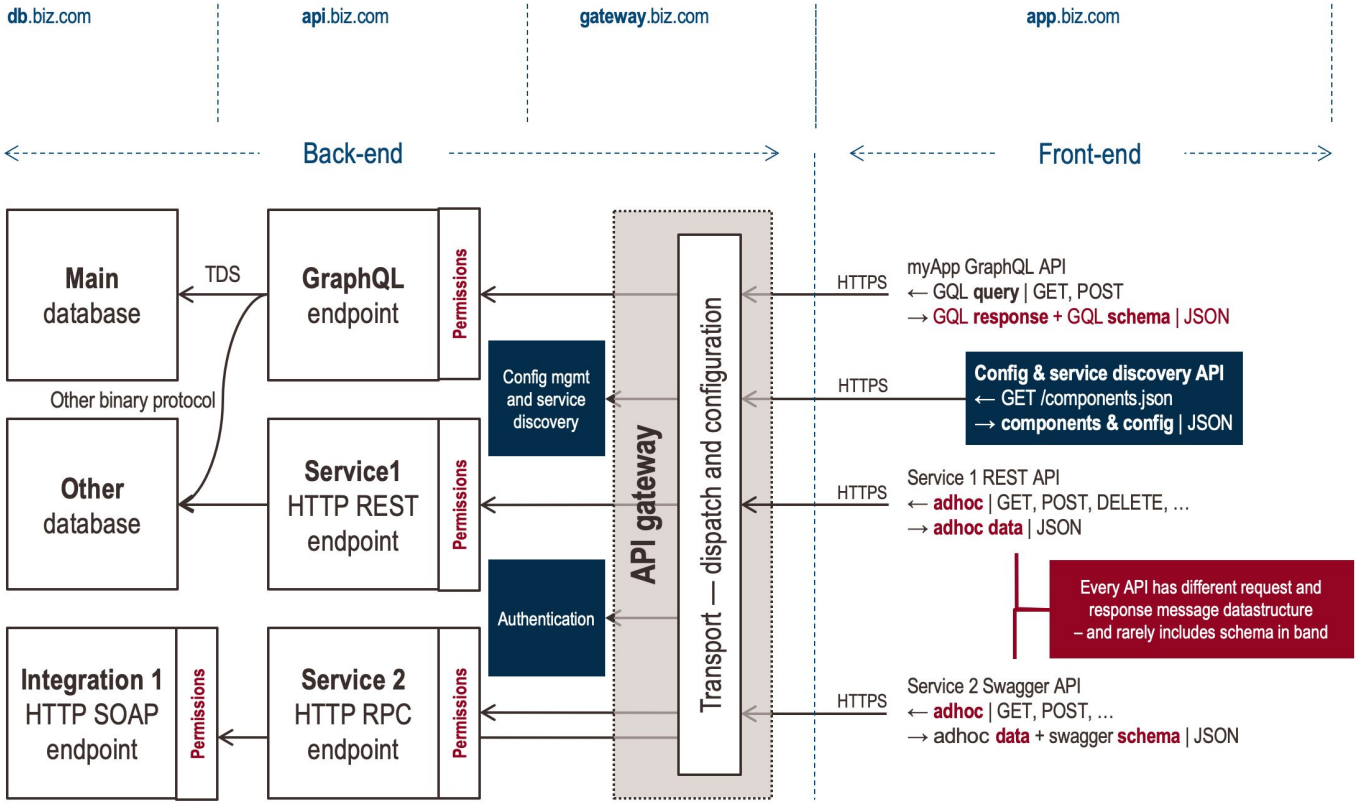
Request federation via REST



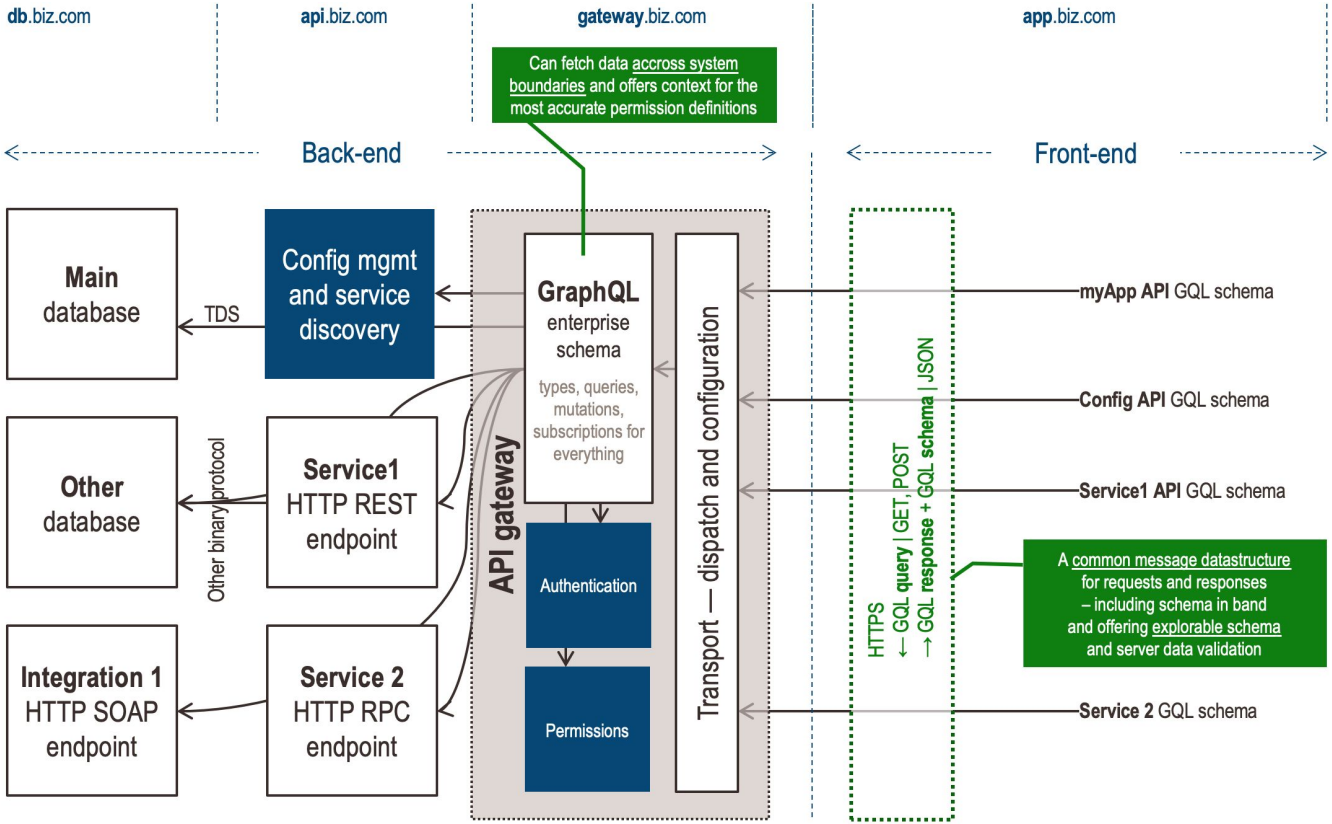
Request federation via *any* protocol



Sample service architecture



Service architecture with simple API gateway



Service architecture with enterprise schema & API gateway

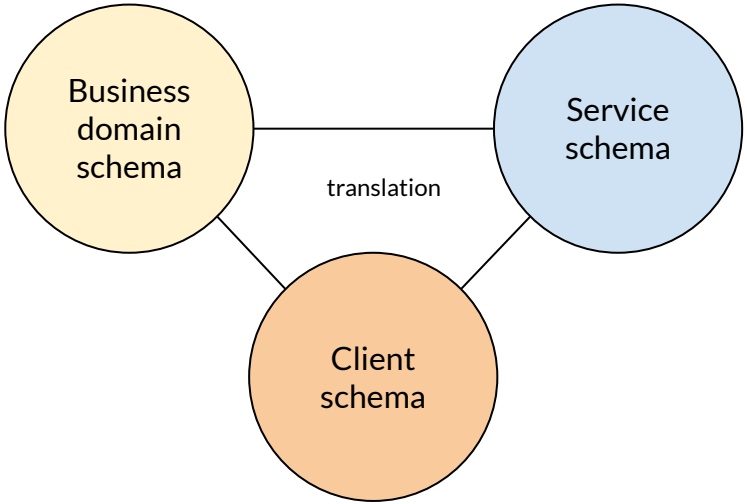
Put on your

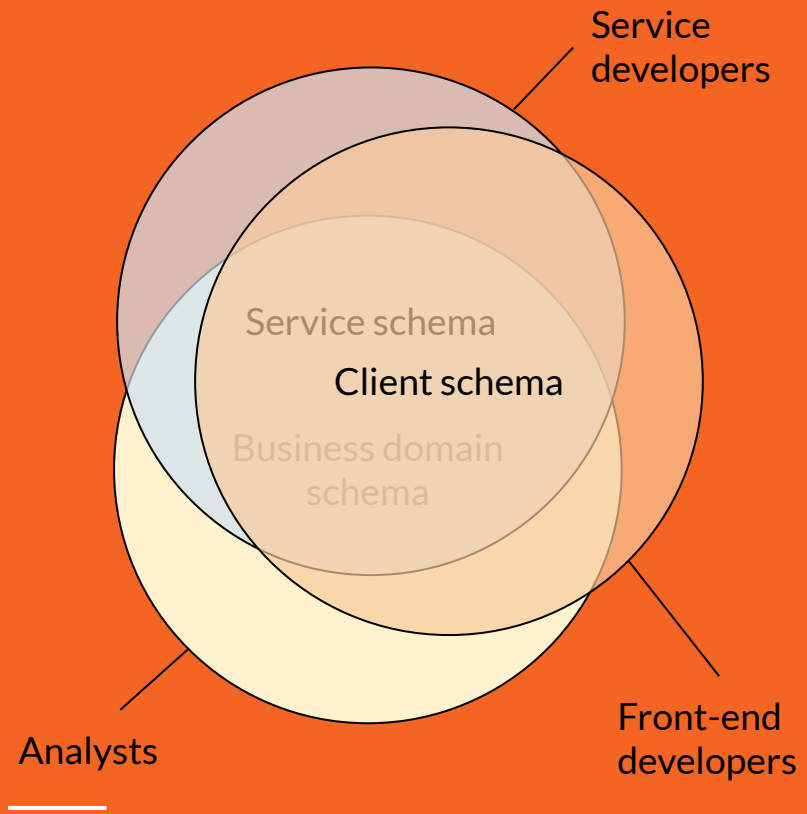
Developer's Hat

Risk & opportunities

Risk & opportunities

Shared language & enterprise schema





Risk & opportunities

Shared language & enterprise schema

Naming things is hard, but important part of building *intuitive APIs*.

Take time to carefully think about what makes sense for your *problem domain* and *users*.

Think of your GraphQL schema as an *expressive shared language* for your *team* and your *users*.

To build a good schema, examine the everyday language you use to describe your business.

Fetch the number of unread emails in my inbox for all my accounts:

```
{
  accounts {
    inbox {
      unreadEmailCount
    }
  }
}
```

Fetch the «preview info» for the first 20 drafts in the main account:

```
{
  mainAccount {
    drafts( first: 20) {
      ...previewInfo
    }
  }
}

fragment previewInfo on Email {
  subject
  bodyPreviewSentence
}
```

Risk & opportunities

Shared language & enterprise schema

Opportunities

- **Shared understanding** of business domaine rules and users
- **Intuitive & discoverable API**
- **Unified schema** enterprise-wide
- **Strong API contracts**

Risks

- **Mirroring legacy database schema** — prefer building a GraphQL schema that describes how clients *use the data* (*data-first* vs *schema-first* approaches)
- **Inability to develop a shared understanding and consensus** of the *business domain rules* and *users*
- Modelling your **entire business domain** in *one sitting* or *without feedback* — build only the part of the schema that you need, for *one scenario at a time*.

Best practices

Best practices

One graph

Your company should have one unified graph, instead of multiple graphs created by each team.

« By having one graph, you maximize the value of GraphQL.

Though there is only one graph, the implementation of that graph should be federated across multiple teams.

- *More data and services can be accessed from a single query;*
- *Code, queries, skills, and experience are portable across teams;*
- *One central catalog of all available data that all graph users can look to;*
- *Implementation cost is minimized, because graph implementation work isn't duplicated;*
- *Central management of the graph — for example, unified access control policies — becomes possible.* »

Best practices

Abstract, Demand-Oriented Schema

The schema should act as an abstraction layer that provides flexibility to consumers while hiding service implementation details.

« The schema should not be tightly coupled either to particular service implementations; or to particular consumers as they exist today. »

*« The standard of a **demand-oriented schema**: a schema focused on providing a great developer experience to an app developer building a new feature against the existing graph.*

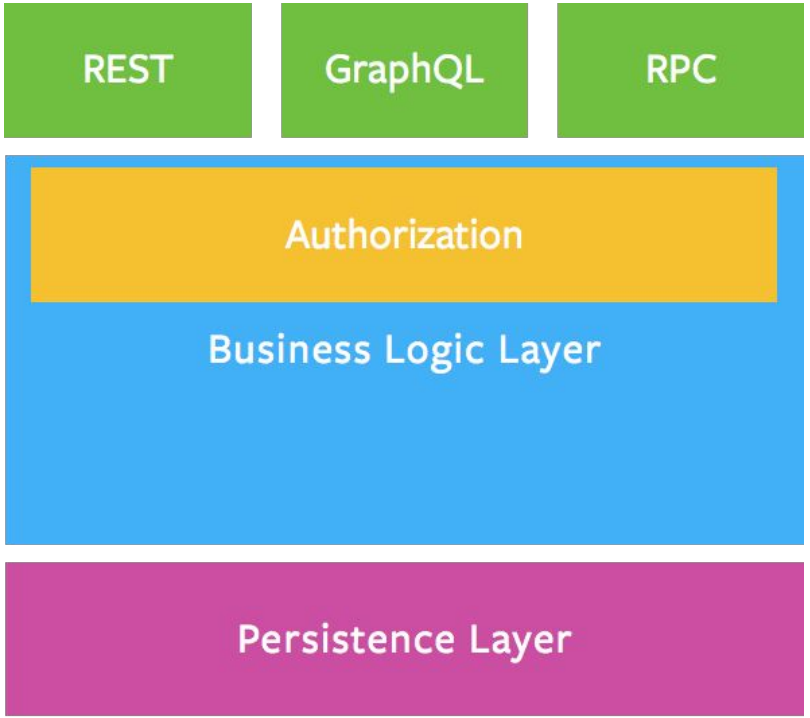
Aiming for this standard will help prevent the graph from becoming coupled to a service implementation that could change in the future, and help increase the reuse value of each field added to the graph. »

Best practices

Business Data Layer

Your business logic layer should act as the single source of truth for enforcing business domain rules

« Where should you define the actual business logic? Where should you perform validation and authorization checks? The answer: inside a dedicated business logic layer. »



Best practices

Authorization

Delegate authorization logic to the business logic layer — have a single source of truth

Authorization is a type of business logic that describes whether a given ‹ **user**; **session**; **context** › *has* permission *to perform an* action *or see a piece of data. For example:*

« Only authors can see their drafts »

Enforcing this kind of behavior should happen in the business logic layer. Although tempting, don't place authorization logic in the GraphQL layer, like so:

```
var postType = new GraphQLObjectType({
  name: 'Post',
  fields: {
    body: {
      type: GraphQLString,
      resolve: (post, args, context, { rootValue }) =>
{
        // return the post body only if the user
        // is the post's author
        if( context.user
           &&( context.user.id === post.authorId)) {
          return post.body;
        }
        return null;
      }
    }
  }
});
```

Best practices

Versioning

Evolve your API without versions

« While there's nothing that prevents a GraphQL service from being versioned just like any other REST API, GraphQL takes a strong opinion on avoiding versioning — by providing the tools for the continuous evolution of a GraphQL schema. »

Why do most APIs version? When there's limited control over the data that's returned from an API endpoint, any change can be considered a breaking change, and breaking changes require a new version. If adding new features to an API requires a new version, then a tradeoff emerges between releasing often and having many incremental versions versus the understandability and maintainability of the API.

In contrast, GraphQL only returns the data that's explicitly requested, so new capabilities can be added via new types and new fields on those types without creating a breaking change. This has led to a common practice of always avoiding breaking changes and serving a versionless API.

Best practices

Server-side Batching and caching

Reduce requests to the various backends via batching and caching

*GraphQL is designed in a way that allows you to write clean code on the server, where **every field** on every type has a focused **single-purpose function** for resolving that value.*

Without additional consideration, a naive GraphQL service could be very « chatty » or repeatedly load data from your databases.

This is commonly solved by a batching technique, where multiple requests for data from a backend are collected over a short period of time and then dispatched in a single request to an underlying database or microservice by using a tool like GraphQL DataLoader.

Best practices

Iteratively Improve Performance

Performance management should be a continuous, data-driven process, adapting smoothly to changing query loads and service implementations.

The data graph layer is the right place to hold the conversation about performance and capacity that always must occur between services teams and the app developers that consume their services.

Rather than optimizing every possible use of the graph, the focus should be on supporting the actual query shapes that are needed in production. Tooling should extract proposed new query shapes and surface them, before they go into production, to all affected service teams with latency requirements and projected query volume. Once the query is in production, its performance should be continuously monitored. If this principle is followed, problems should be easy to track back to the service that is not behaving as expected.

Best practices

Structured Logging

Capture structured logs of all graph operations and leverage them as the primary tool for understanding graph usage

***Business information**: who performed the operation, what was accessed or changed, which feature of which app built by which developer, whether it succeeded, how it performed. **Technical information**: which backend services were contacted, how each service contributed to latency, whether caches were used?*

Traces for all graph operations should be collected in one central place, so that there is one authoritative stream of traces. They can be used for a wide range of purposes:

- *Understanding whether a deprecated field can be removed, or if not, the specific clients that are still accessing it and how important they are;*
- *Providing an authoritative audit trail showing which users have accessed a particular record;*
- *Generating invoices for partners based on API usage.*

Benefits of GraphQL

As a conclusion

Benefits of GraphQL

« An Overview of GraphQL » by **William Lyon**
Developer Relations Engineer, Neo4j

Schema/type system — « GraphQL requires a user-defined schema built using a strict type system. This schema acts as a blueprint for the data handled by the API, allowing developers to know exactly what kind of data is available and in what format. Developer tools can take advantage of the schema through the use of introspection, enabling documentation, query auto-complete and mocking tools. »

Request only data needed — « Since GraphQL queries can define what data exactly is needed at query time, all the data required to render an application view can be requested in a single request, reducing the number of roundtrip network requests and with that, the application latency. »

Wrapping existing data services — « GraphQL can work with any database or data layer, allowing for a single GraphQL service to fetch data from multiple sources in a single request served by the composed API. »

Further reading & notes

Discovering · Sources of this talk

An Overview of GraphQL · Neo4j, William Lyon
dzone.com/refcardz/an-overview-of-graphql

Evolving the Graph · Coursera, Jon Wong · 28.08.2019
youtube.com/watch?v=fmsDlaKTJZs
medium.com/coursera-engineering/evolving-the-graph-4c587a4ad9a8

GraphQL A query language for your API · Homepage
graphql.org

Learning · Essential Docs

GraphQL · Learn, community, foundation
graphql.org/learn/

GraphQL Specification · Type system, query syntax, validation and introspection
github.com/graphql/graphql-spec

GraphQL Implementations · C#, Go, Java, Kotlin, Scala, Clojure, JavaScript, Python, Ruby, Groovy, PHP & more
graphql.org/code/

Learning · Essential Docs

How-to GraphQL · Fullstack Tutorials for GraphQL
howtographql.com

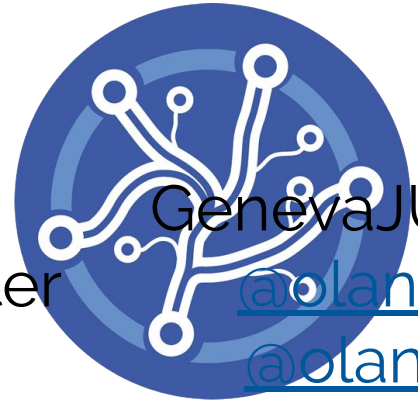
Anatomy of a GraphQL Query · GraphQL 101 vocabulary
blog.apollographql.com/the-anatomy-of-a-graphql-query

Principled GraphQL · Best practices for creating, maintaining, and operating a data graph
principledgraphql.com

Thank you
for your attention


Your questions
are very welcome

Your **feedback** is appreciated



Leave a comment on GenevaJUG
or ping me on Twitter @olange
or on LinkedIn @olange

or join our bit.ly/gql-primer-channel on Slack

Gōng-fu I/O

Hack sions

Come, practice and earn

every wednesday n30

Presentation (in french)

http://bit.ly/gfio-10min