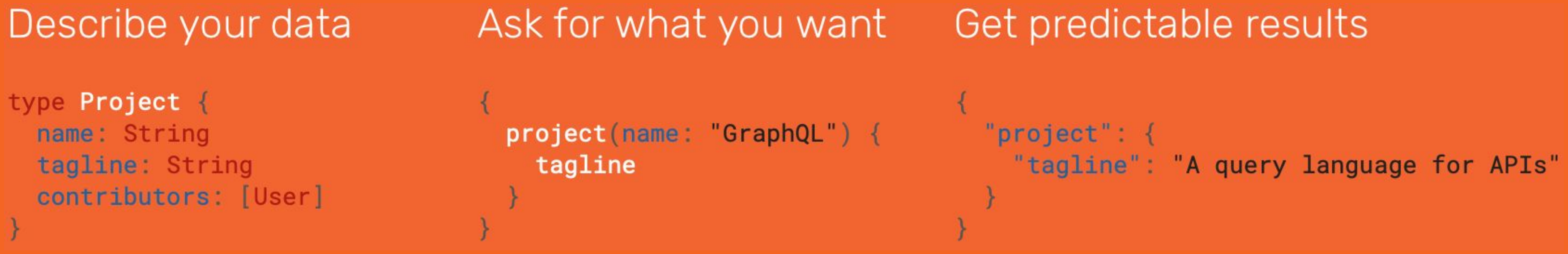
GraphQL
a data language
and new paradigm
for building APIs

An overview for software { architects, developers }

ISO · Tech talks · 10.10.2019 · bit.ly/gql-overview

Olivier Lange · twitter/github @olange

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

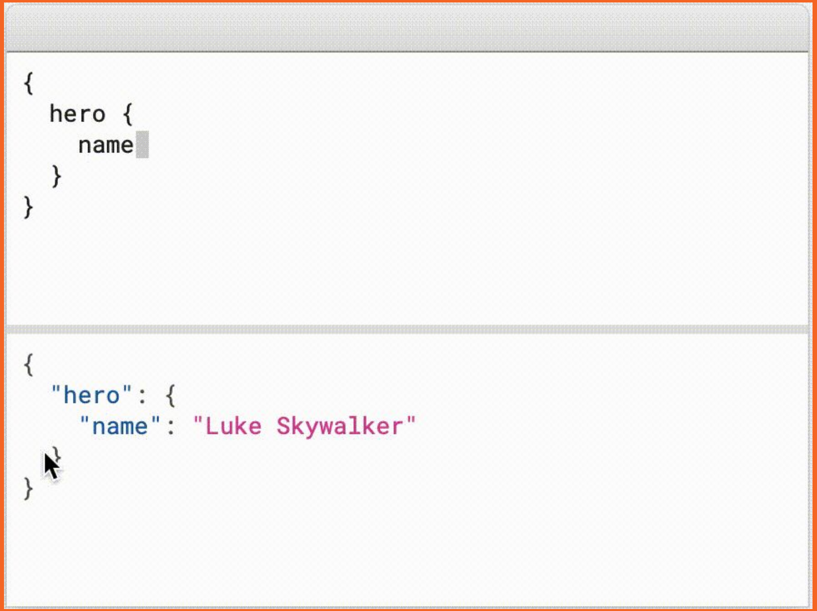
```
{
  "project": {
    "tagline": "A query language for APIs"
  }
}
```

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. »



```
{
  hero {
    name
  }
}
```

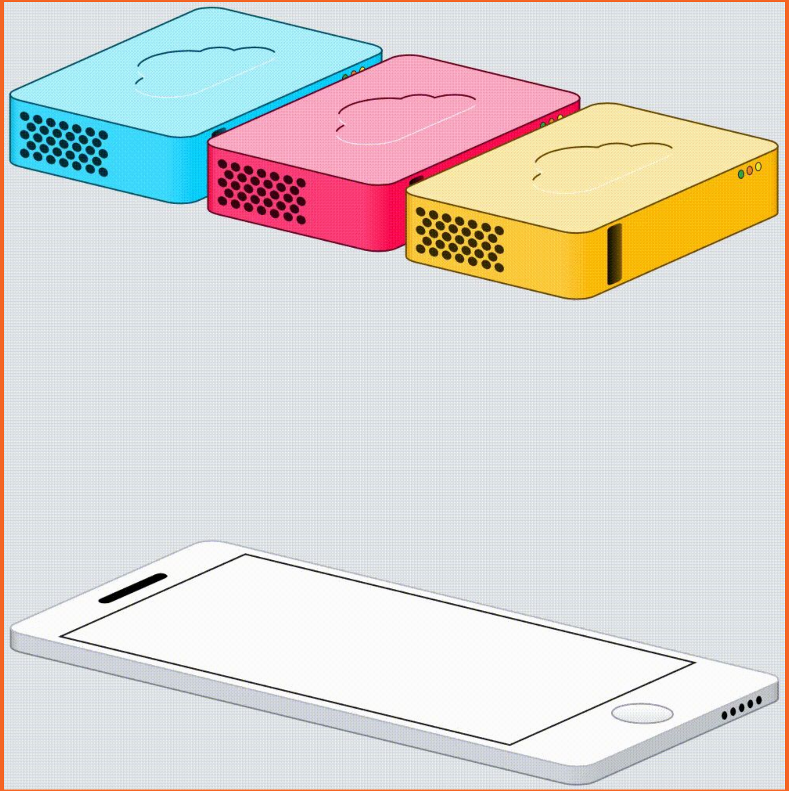
```
{
  "hero": {
    "name": "Luke Skywalker"
  }
}
```

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. »



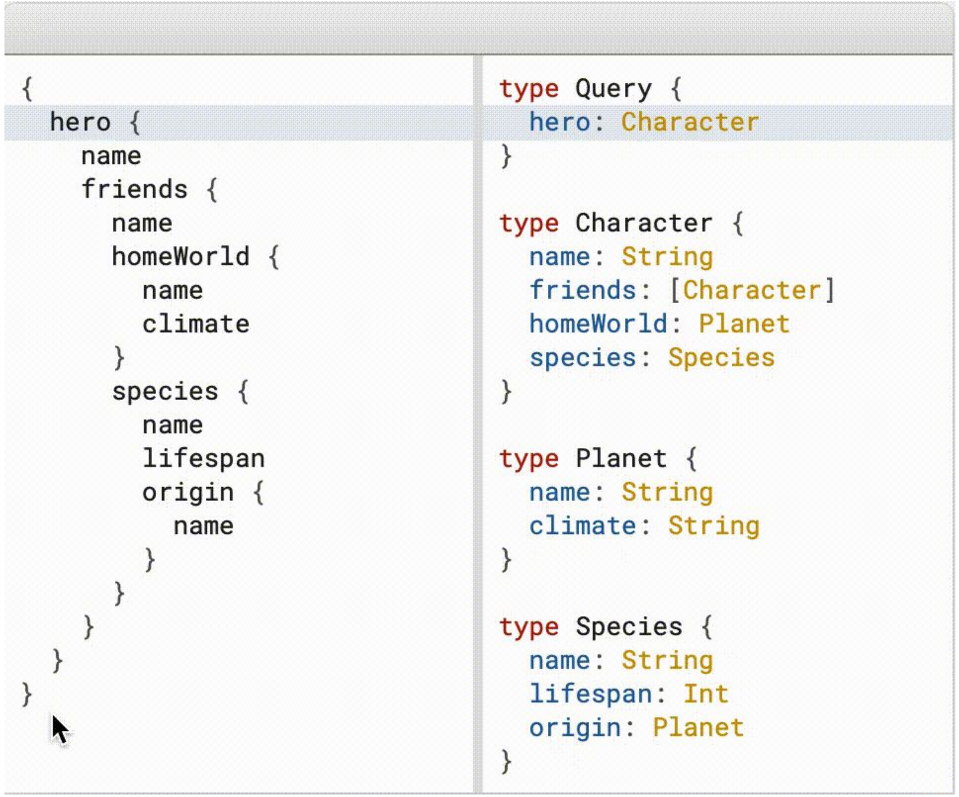
What is GraphQL?

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
        }
      }
    }
  }
}
```

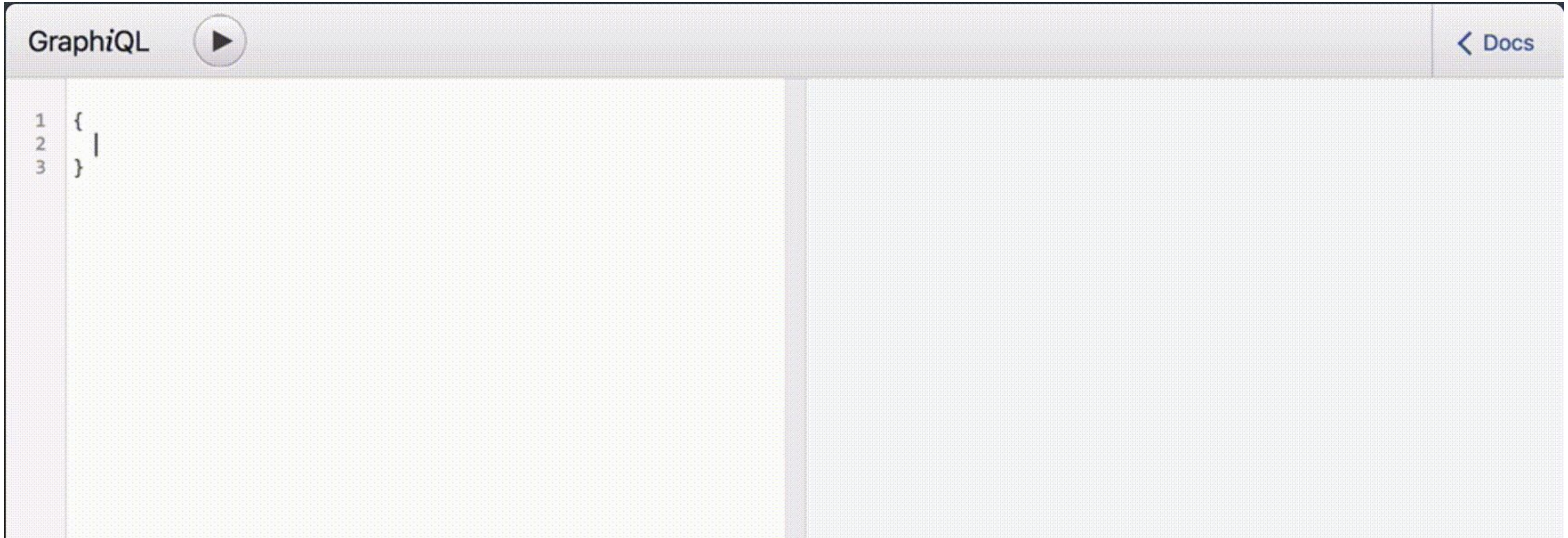
```
type Query {
  hero: Character
}

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*.

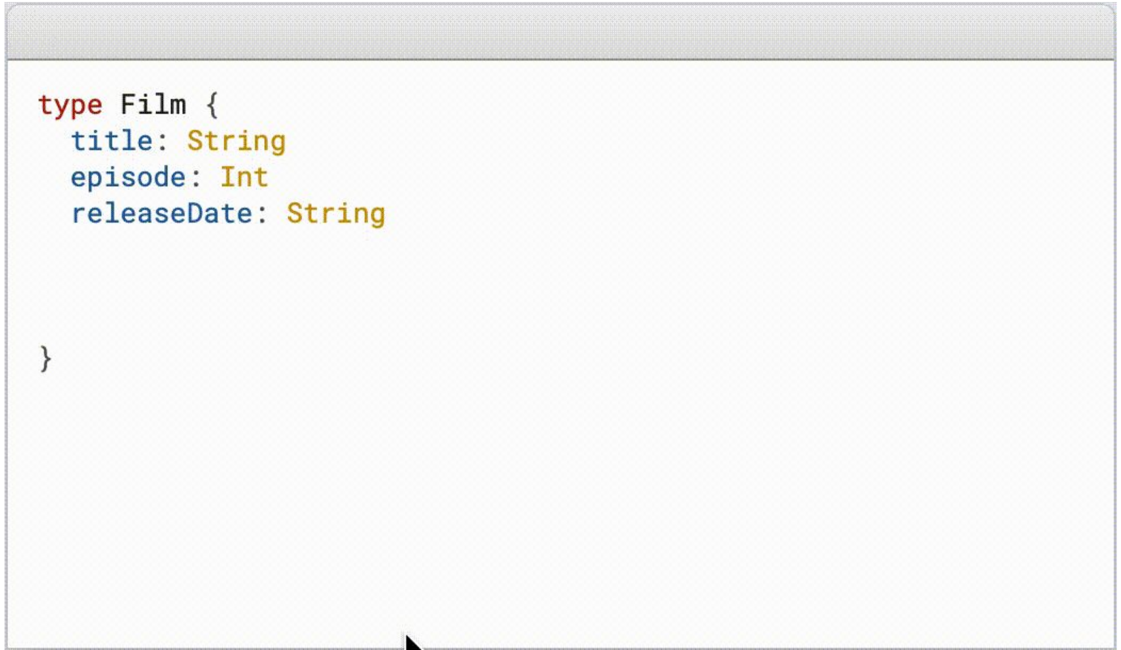
Hierarchical — nested fields let you query for and receive only the data you specify in a single round trip.



```
{
  |
}
```

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.

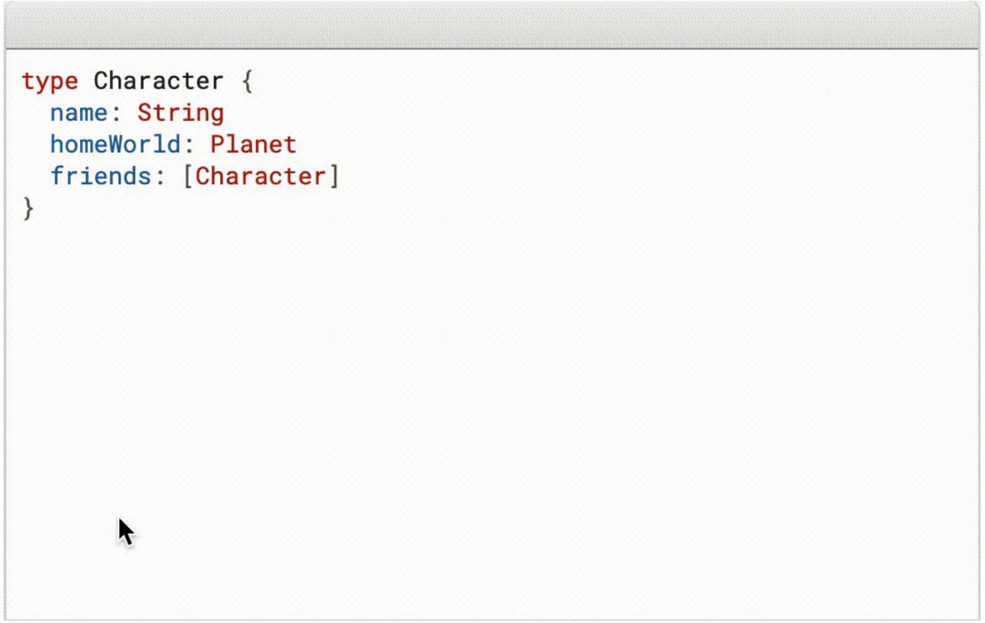
What is GraphQL?

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.* »



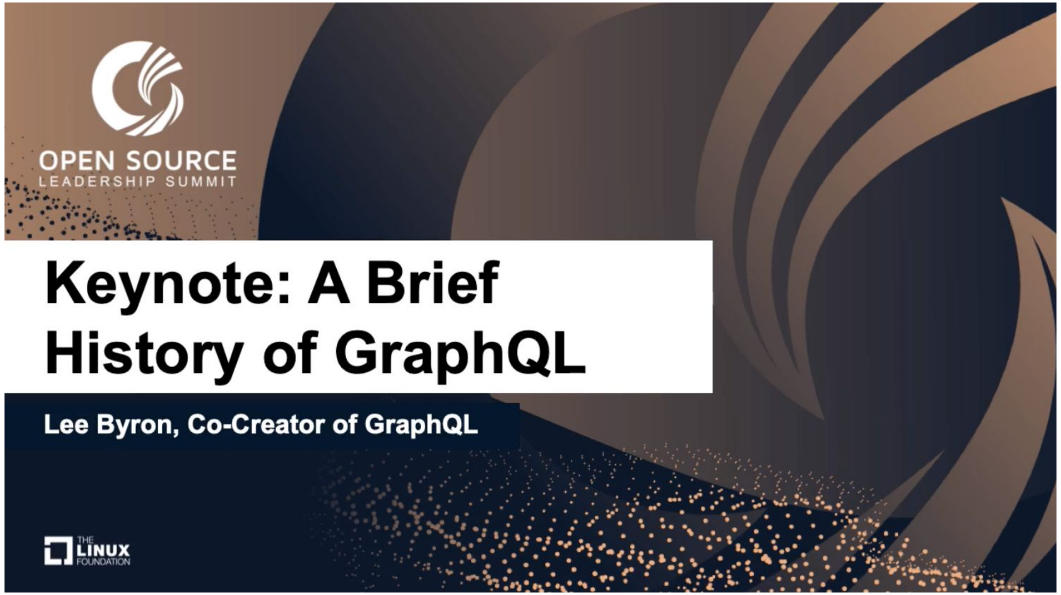
```
type Character {
  name: String
  homeWorld: Planet
  friends: [Character]
}
```

A brief history of GraphQL

Brief history

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

YouTube · video 11mn. · 15.03.2019


OPEN SOURCE
LEADERSHIP SUMMIT
Keynote: A Brief History of GraphQL
Lee Byron, Co-Creator of GraphQL
THE LINUX FOUNDATION

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

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. »

8base APOLLO aws Butterfly Dgraph

ELEMENTL expedia group facebook Gatsby GraphZen GraphQL for .NET

HASURA IBM intuit powering prosperity neo4j NOVVUM

pipefy Prisma SALSIFY shopify solo.io

Background

Who's using GraphQL?

Seriously!

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

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

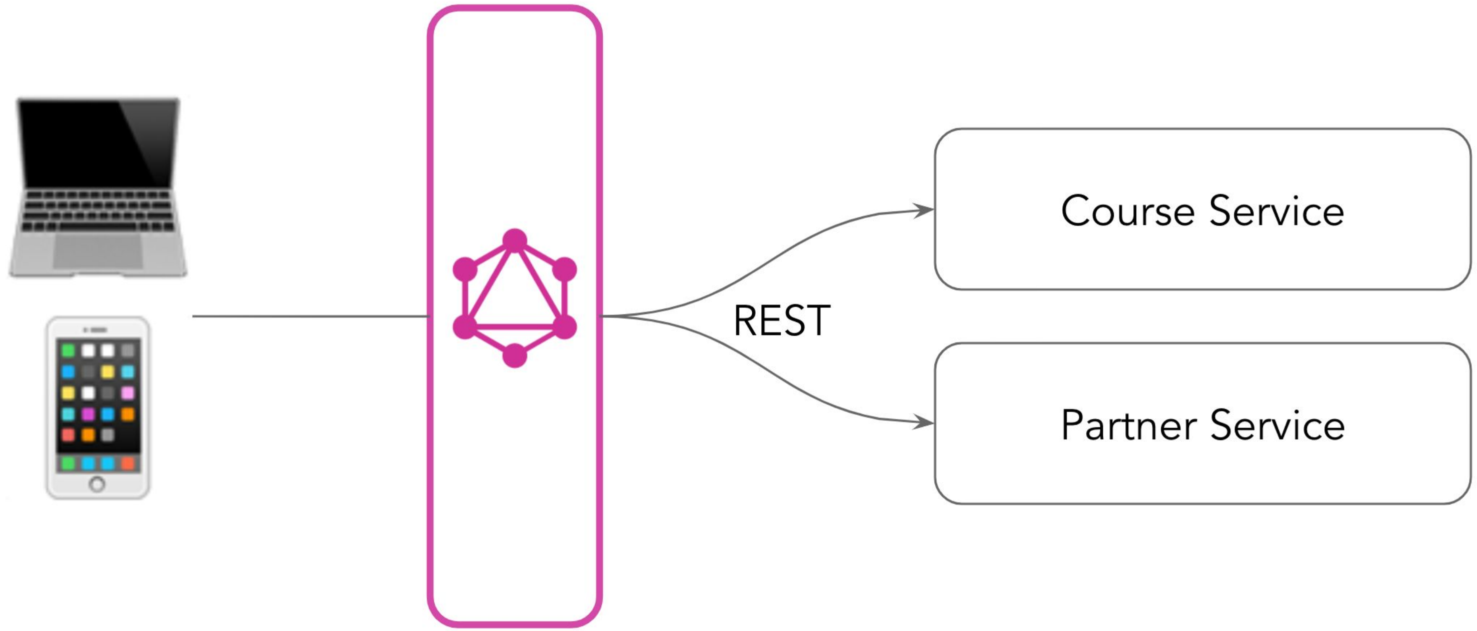
Who's using GraphQL?

20 minutes ALLOCINE ArangoDB ATLASSIAN

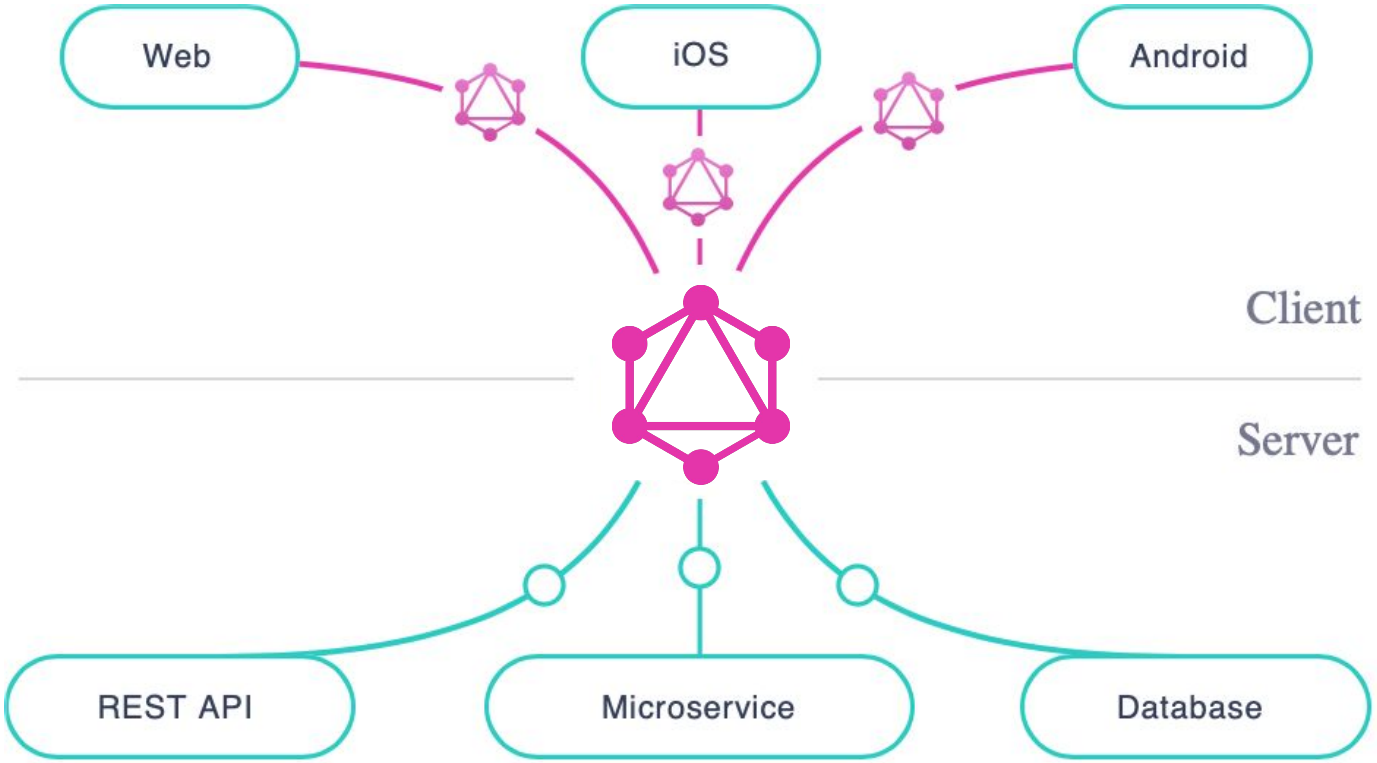
HACKAGES KLM METEOR MyHeritage

neo4j PayPal sky yelp

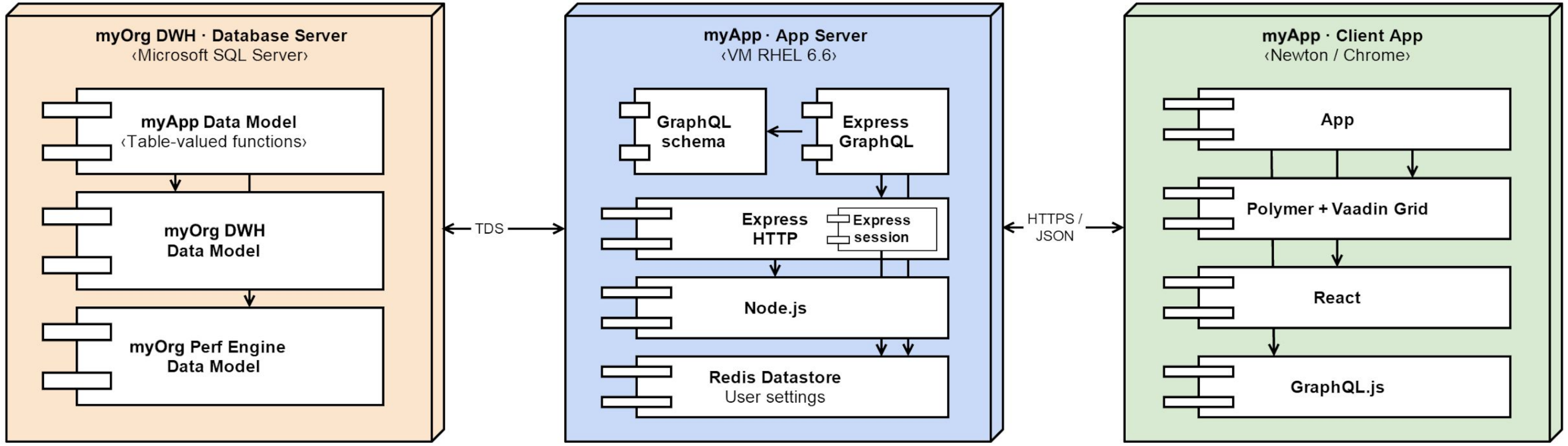
Put on your **Architect's Hat**



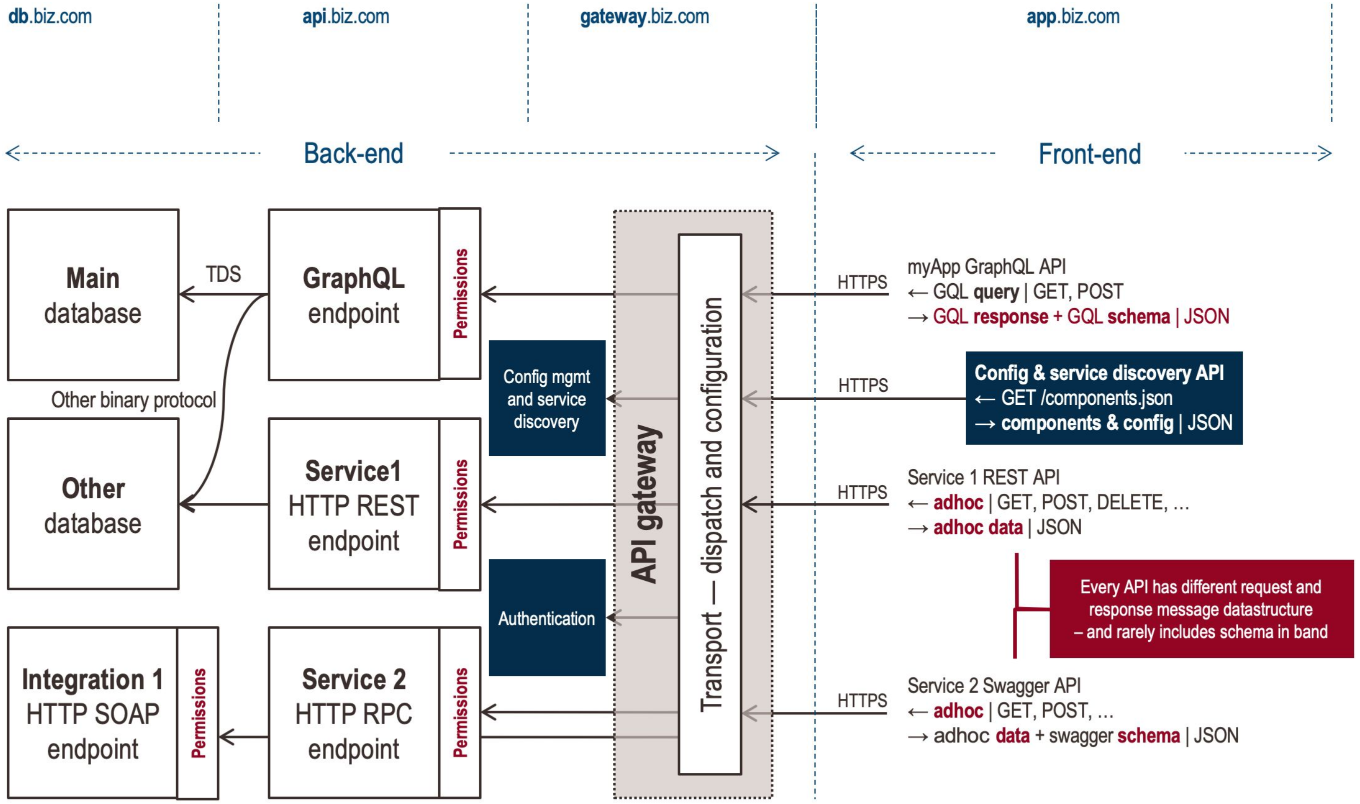
Request federation via REST



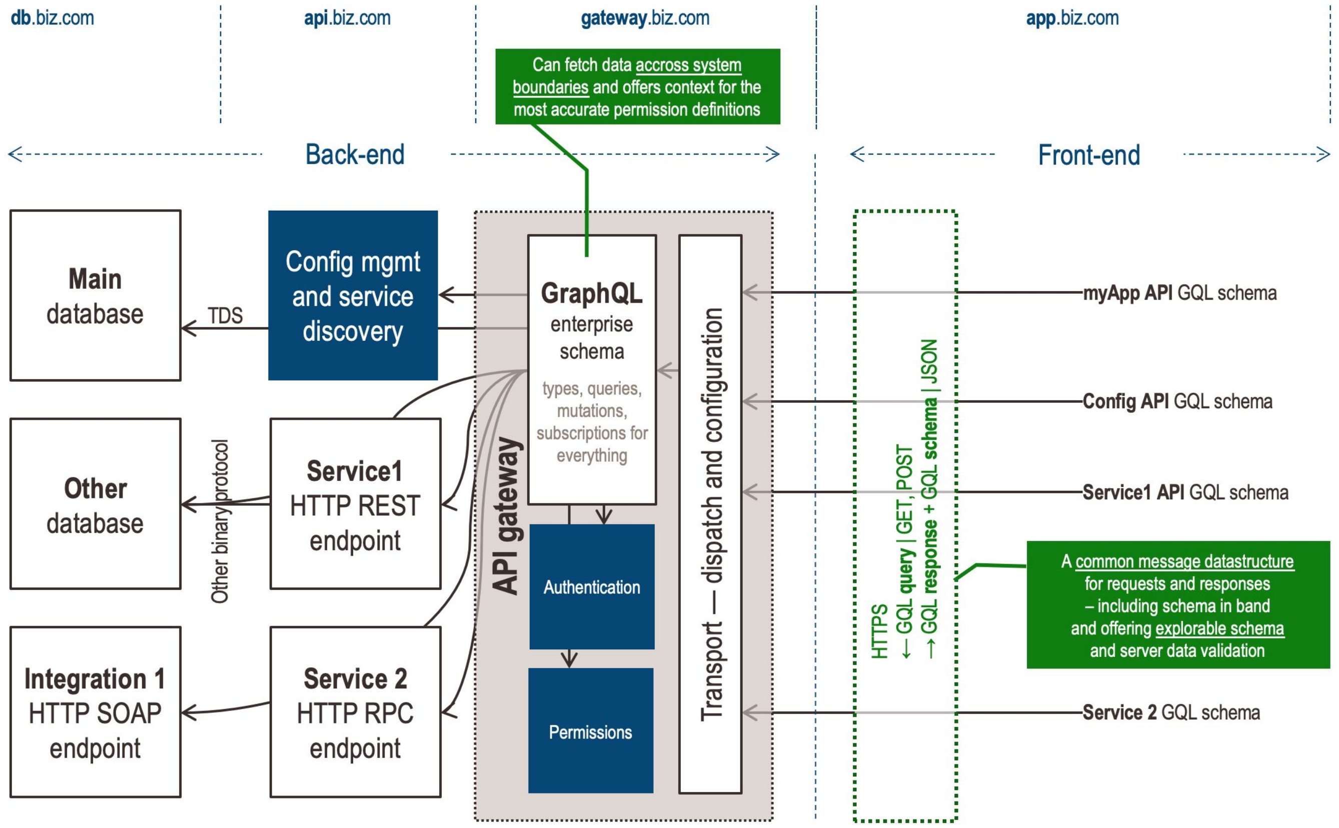
Request federation via *any* protocol



Sample service architecture


db.biz.com
api.biz.com
gateway.biz.com
app.biz.com
Back-end
Front-end
Main database
TDS
GraphQL endpoint
Permissions
Other binary protocol
Other database
Service1 HTTP REST endpoint
Permissions
Integration 1 HTTP SOAP endpoint
Permissions
Service 2 HTTP RPC endpoint
Permissions
Config mgmt and service discovery
Authentication
API gateway
Transport — dispatch and configuration
HTTPS
myApp GraphQL API
← GQL query | GET, POST
→ GQL response + GQL schema | JSON
HTTPS
Config & service discovery API
← GET /components.json
→ components & config | JSON
HTTPS
Service 1 REST API
← adhoc | GET, POST, DELETE, …
→ adhoc data | JSON
Every API has different request and response message datastructure – and rarely includes schema in band
HTTPS
Service 2 Swagger API
← adhoc | GET, POST, …
→ adhoc data + swagger schema | JSON

Service architecture with simple API gateway


db.biz.com
api.biz.com
gateway.biz.com
app.biz.com
Can fetch data accross system boundaries and offers context for the most accurate permission definitions
Back-end
Front-end
Main database
TDS
Config mgmt and service discovery
GraphQL enterprise schema
types, queries, mutations, subscriptions for everything
Transport — dispatch and configuration
API gateway
Other database
Other binary protocol
Service1 HTTP REST endpoint
Authentication
Permissions
Integration 1 HTTP SOAP endpoint
Service 2 HTTP RPC endpoint
HTTPS
← GQL query | GET, POST
→ GQL response + GQL schema | JSON
myApp API GQL schema
Config API GQL schema
Service1 API GQL schema
A common message datastructure for requests and responses – including schema in band and offering explorable schema and server data validation
Service 2 GQL schema

Service architecture with enterprise schema & API gateway

Put on your

Developer's Hat

Risk & opportunities

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

Think in graphs

With GraphQL, you model your business domain as a graph

« Graphs are powerful tools for modeling many real-world phenomena, because they resemble our natural mental models and verbal descriptions of the underlying process. »

With GraphQL, you model your business domain *as a* graph *by defining a* **schema**. *Within your* schema, *you define different* types of nodes *and how they* connect/relate *to one another.*

On the ***client****, this creates a pattern similar to OO-programming:* types *that reference other* types.

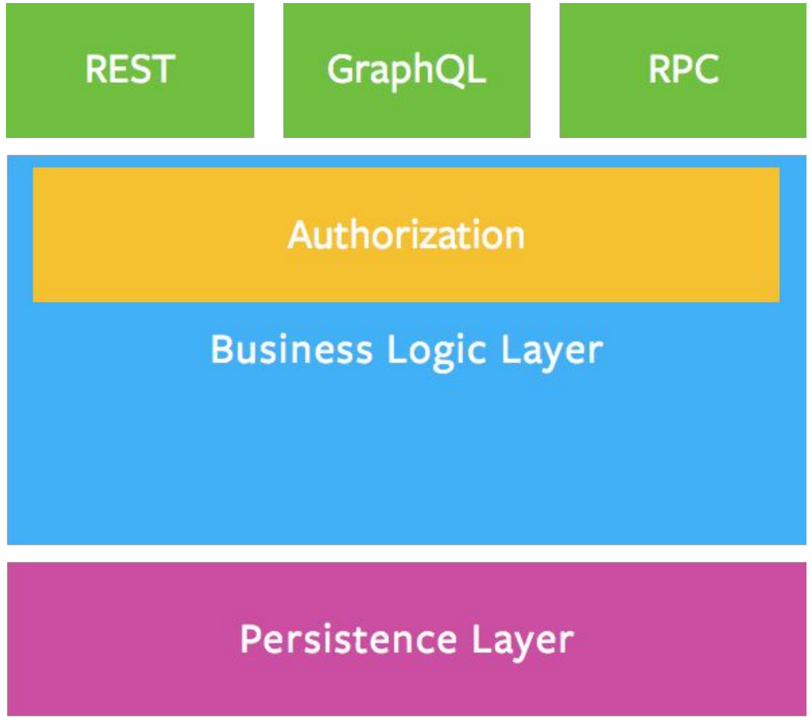
On the ***server****, since GraphQL only defines the* interface, *you have the freedom to use it with* **any backend** *(new or legacy).*

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.

Sources & further reading

Discovering and learning

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

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

Essential documentation

GraphQL · Get started, learn, community, foundation
https://graphql.org

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

GraphQL Implementations · C#, Go, Java, Scala, Clojure, JS, Python, Ruby & more
https://graphql.org/code/

Thank you

for your attention.

Your ns

& discussion

are very welcome