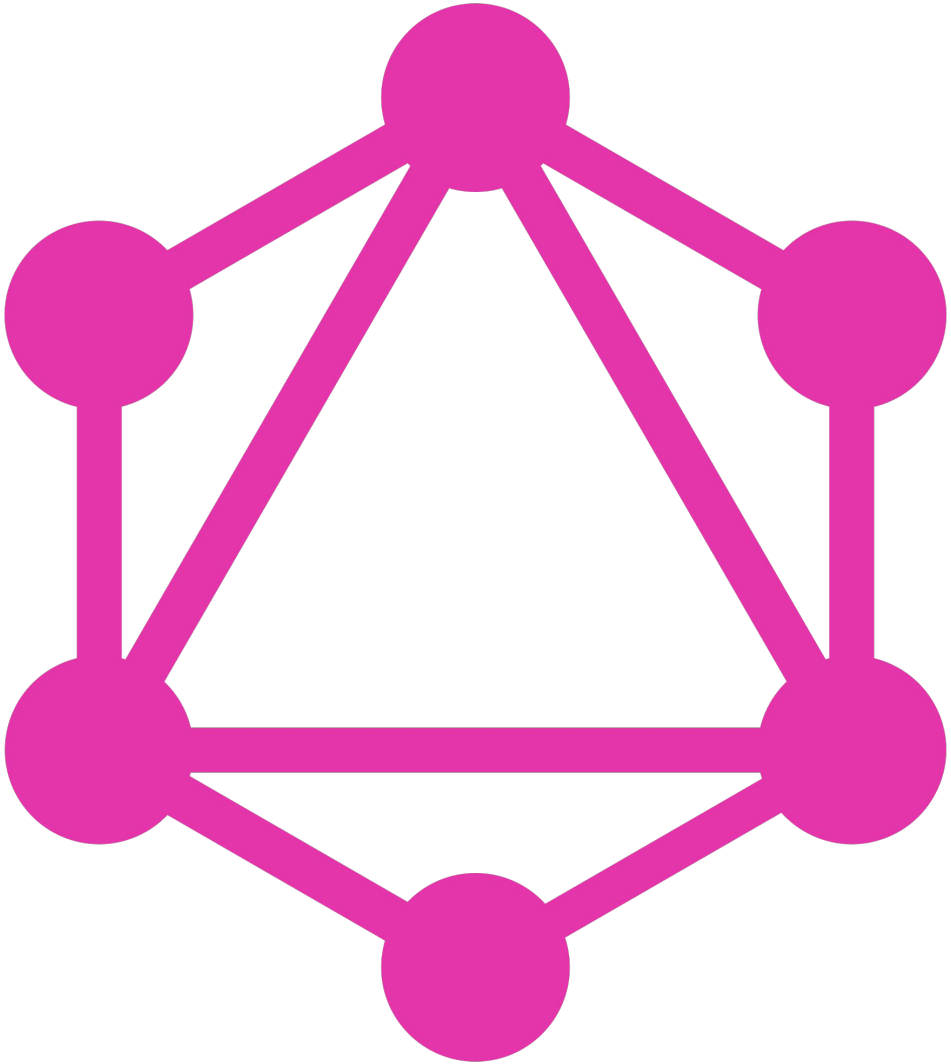
GraphQL

25 January 2019

MoHo Khaleqi



What problems is GraphQL trying to solve?

One endpoint for all

Having single endpoints for each entity means that clients must make several calls to populate a UI that displays multiple entities.

Example:

Request from outside

- 205ms
  - 200ms network
  - 5ms data)

---

3 request: 615ms

Request internally

- 6ms
  - 1ms network
  - 5ms data

---

3 request: 18ms

One endpoint for all - GraphQL example

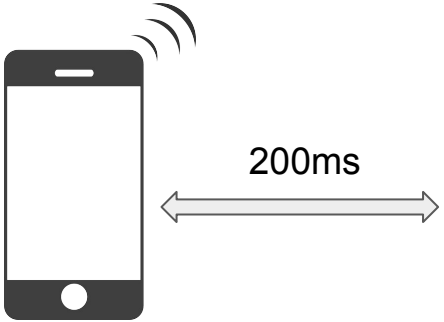
GraphQL

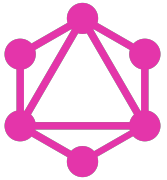
1 request

- 200ms latency to GraphQL server
- A internal request => 6ms

Fetch user with recent posts and friends list

Total: 218ms





6ms to fetch user
6ms to fetch recent posts
6ms to fetch list of friends

No more over/under fetching

- Overfetching
  - Downloading unnecessary data
- Underfetching
  - An endpoint doesn’t return enough of right information
  - Need to send multiple request (n+1 request problem)

Rapid product Iterations

- Structure endpoints according to client’s data need
- No Need to adjust API when product requirements and design change
- Faster feedback cycle and product iterations

Insight Analytics

- Transparent understanding about what data is read by clients
- Enables evolving API and depracting unused/unneeded api features
- Low level performance monitoring by checking resolvers

Benefits of Schema and types

- Strong typed system
- Schema serves as contract between client and server
- Frontend and backend team can work independently from each other

What is GraphQL

What is GraphQL?

- Originally created at Facebook in 2012 and being used in their mobile apps
- First time presented publically at React.js conf 2015
- A query language and execution engine
- A new API standard
- Enables declarative data fetching - Define what exactly needed
- GraphQL server expose a single endpoint and response to queries.

What is GraphQL?

In a nutshell, GraphQL is a query language that decreases roundtrips between clients and APIs by packaging requests and responses, and can help slim down unnecessary data from over-fetching.

It’s just a specification on how an API should work.

GraphQL isn't tied to any specific database or storage engine and is instead backed by your existing code and data.

The GraphQL is:

- A Specification
  - The spec determines the validity of the schema on the API server. The schema determines the validity of client calls.

- Client-specified queries
  - A GraphQL query, on the other hand, returns exactly what a client asks for and no more.

The GraphQL is:

- Hierarchical.
  - The shape of a GraphQL call mirrors the shape of the JSON data it returns. Nested fields let you query for and receive only the data you specify in a single round trip.

Request:

```
{
  user(id: 13) {
    name
  }
}
```

Response in JSON

```
{
  "user": {
    "name": "MoHo"
  }
}
```

The GraphQL is:

- An application layer
  - GraphQL is not a storage model or a 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.
- Strongly typed
- Introspective
  - A GraphQL server type system must be queryable by the GraphQL language itself, as will be described in this specification

Programming language support

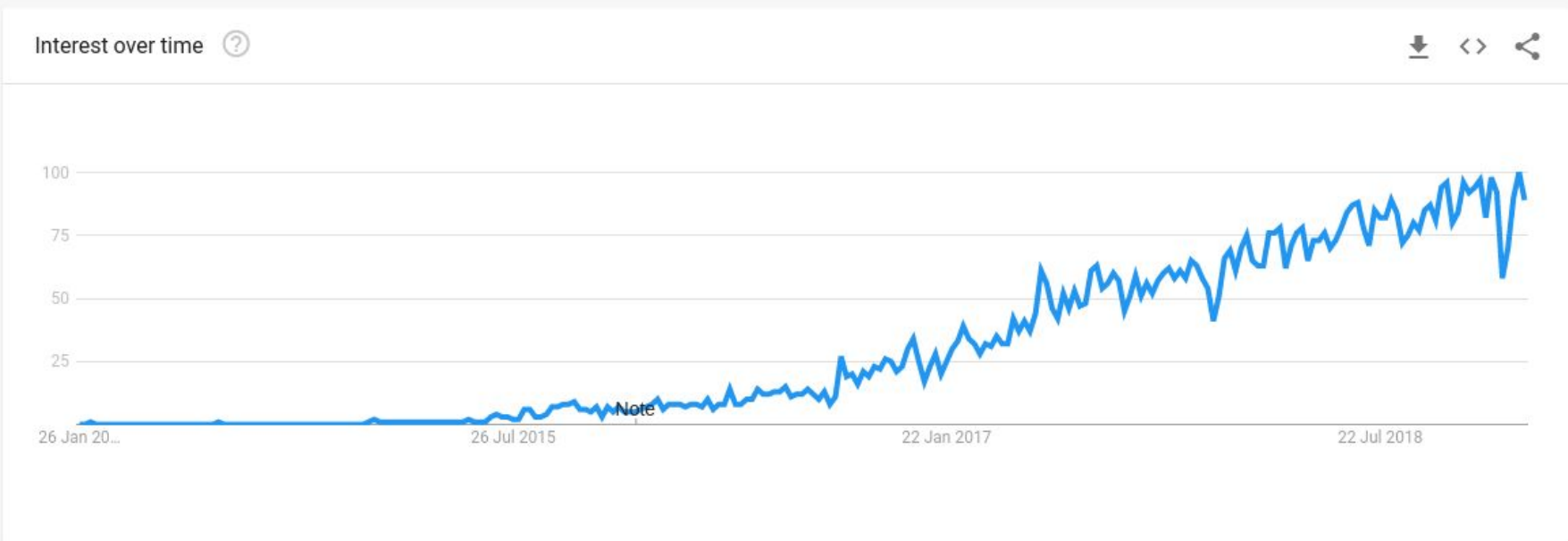
GraphQL does not mandate a particular programming language or storage system for application servers that implement it.

- C# / .NET
- Clojure
- Elixir
- Erlang
- Go
- Groovy
- Java
- JavaScript
- PHP
- Python
- Scala
- Ruby

What is GraphQL not?

- GraphQL is not a programming language capable of arbitrary computation.
- It is not storage either.
- It’s not a framework
- It’s not a library.

Google Trends



Who’s using GraphQL?

It’s a mature and production-tested technology

1stdibs
20 minutes
adayroi.com
ALEMBIC
ALLOCINE
AlphaSights
Amplitude
Appier
ArangoDB
ARTSY
ATLASSIAN
attendify
bazinga!
BlenderBottle
Bright
Buildkite
bynder
cheddar
CircleHD
CLOVERLEAF
ClubMed
colectica
compara online
Conduit
coursera
credit karma
curio
dailymotion
digitransit
Drift
DueDil
eastview christian church
easy carros
ediket
EXPERT360
Fairfax Media
gentux
GetNinjas
Goalify
GRAPHCMS
GRAPHCOOL
HACKAGES
HASURA
HSL HRT
HIJUP
HOUSING ANYWHERE
hudl
Icon Systems
idobata
IndonesiaX
inerva
intuit
Jusbrasil
KLM
leanIX
LEGENDS OF LEARNING
lelivrescolaire.fr
Lets.events
MAKE SCHOOL
MEDALLIA
METEOR
metric:ai
Mixcloud
MyHeritage
NEWS
neo4j
NEWSPRING CHURCH
NINGENSOFT
Nova-Ideo
OK GROW!
OVOS
PayPal
PERSADO[GO]
Product Hunt
PROJECT SEPTEMBER
PROTEL
Quri
REDBUBBLE
Reindex
Restorando
SALE STOCK
scaphold
SERVERLESS
sky
SKYARCH
SMARKETS
stackshare
startups.co
stem
swapcard
SIZIGI
taller
Teachers Pay Teachers
teselagen
THE HUNT
Trove
UC Trend
Unigraph:io
universe
UST Global
vanila.
waldo
wayfair
whitescape
wirtualna polska
wishlife
WorkflowGen
WOW
yelp
zlyde
zzish

Core Concepts

Terminology

- Type
  - As the name says Type is the type of Object/Field we're creating or accessing
    - GraphQLObjectType
    - GraphQLID
    - GraphQLInt
    - GraphQLString
    - GraphQLFloat
    - GraphQLList
    - GraphQLNonNull
  - And more http://graphql.org/learn/schema/

The Schema Definition Language (SDL)

```
type Person {
  name: String!
  age: Int!
}
```

```
type Post {
  title: String!
}
```

The Schema Definition Language (SDL)

Adding Relationship

```
type Person {
  name: String!
  age: Int!
  posts: [Post!]!
}
```

```
type Post {
  title: String!
  author: Person!
}
```

Terminology

- Query
  - It’s like a GET request, Everytime we want to access the data for reading, We make use of Queries.

```
{
  allPersons {
    name
  }
}
```

```
{
  "allPersons": [
    { "name": "MoHo" },
    { "name": "JooJoo" },
    { "name": "Ieva" }
  ]
}
```

Terminology

- Query
  - It’s like a GET request, Everytime we want to access the data for reading, We make use of Queries.

```
{
  allPersons(last: 2) {
    name
    age
  }
}
```

```
{
  "allPersons": [
    { "name": "JooJoo", "age": 26 },
    { "name": "Ieva", , "age": 20 }
  ]
}
```

Terminology

- Query
  - It’s like a GET request, Everytime we want to access the data for reading, We make use of Queries.

```
{
  allPersons(first: 2) {
    name
    posts(first: 2) {
      title
    }
  }
}
```

```
"allPersons": [
      {
        "name": "MoHo",
         "posts": [
          {
            "title": "Learn GraphQL today"
          },
          {
            "title": "Do you react?"
          }
        ]
      },
      {
        "name": "JooJoo",

        "posts": []
      }
    ]
  }
}
```

Terminology

- Mutation
  - Just like queries, Mutation is used whenever we want to manipulate the data, This is used for *Create*, *Update* and *Delete* operations

```
mutation {
  createPerson(name: "Michael", age: 24) {
    name
    age
  }
}
```

Terminology

Query or Mutation act like a proxy. They process the incoming GraphQL request and call the *resolver* method with passing all parameters.

Validation are done in Query and Mutation.

Terminology

- Resolver
  - We use resolve for fetching and manipulating the data.
  - We can fetch the data from Database, API, File, or any datasource we can imagine/have access to, GraphQL doesn't restrict us.
- Subscriptions
  - Subscriptions are like queries, but everytime the data changes the query is run and new response is sent back to all* connected clients. Realtime update

Terminology

- Schema
  - Here we connect everything together.
  - A schema defines a GraphQL API's type system. It describes the complete set of possible data.
  - Calls from the client are validated and executed against the schema.
  - A schema resides on the GraphQL API server.

```
type Query {
  allPersons(last: Int): [Person!]!
}

type Mutation {
  createPerson(name: String!, age: Int!):
Person!
}

type Subscription {
  newPerson: Person!
}

type Person {
  name: String!
  age: Int!
  posts: [Post!]!
}

type Post {
  title: String!
  author: Person!
}
```

Example

Blogging App in REST

MoHo

MoHo's posts:

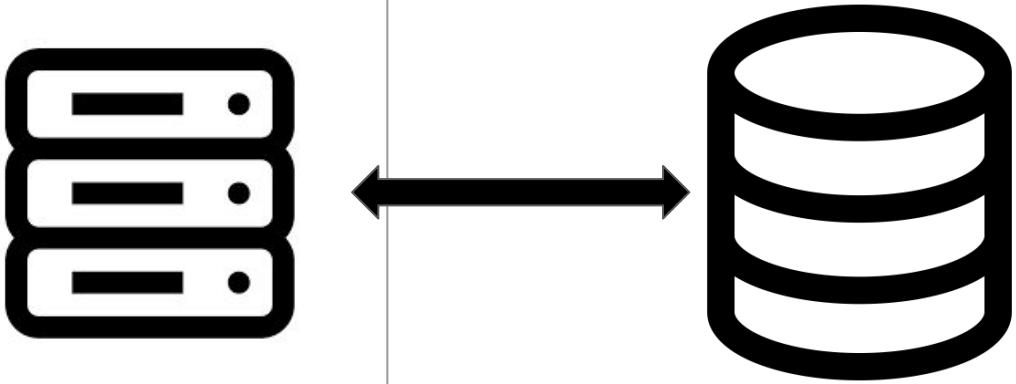
- Learn about GraphQL today
- Do you React?
- What's Next.js?
- TypeScript could help maybe if...

Last 3 Followers

- Morgan
- JoJo
- Ieva

Blogging App in REST

- `/user/:id`
- `/user/:id/posts`
- `/user/:id/followers`



Blogging App in REST

MoHo

MoHo's posts:

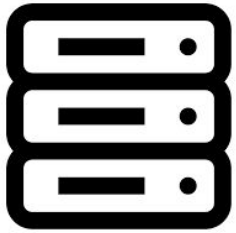
- Learn about GraphQL today
- Do you React?
- What's Next.js?
- TypeScript could help maybe if...

Last 3 Followers

- Morgan
- JoJo
- Ieva

Fetch User

- **/user/:id**
- /user/:id/posts
- /user/:id/followers



HTTP GET

```
{
    "User": {
        "Id": "ebg14rh7kfhd"
        "Name": "MoHo"
        "Address": "Leatherhead"
        "Birthday": "15.06.1988"
        ...
    }
}
```

Blogging App in REST

MoHo

MoHo's posts:

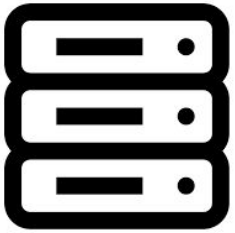
- Learn about GraphQL today
- Do you React?
- What's Next.js?
- TypeScript could help maybe if...

Last 3 Followers

- Morgan
- JoJo
- Ieva

Fetch Posts

- /user/:id
- **/user/:id/posts**
- /user/:id/followers



HTTP GET

```
{
    "Posts": [{
        "Id": "jsdfh342hdf"
        "title": "Learn about GraphQL today"
        "content": "Lorem ..."
        "comments": [{...}]
        "createdAt": "25012019"
        ...
    },
    {post 2}, {post 3}, {...}
    ]
}
```

Blogging App in REST

MoHo

MoHo's posts:

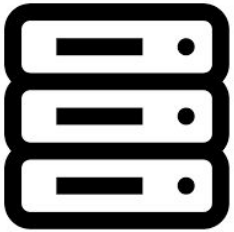
- Learn about GraphQL today
- Do you React?
- What's Next.js?
- TypeScript could help maybe if...

Last 3 Followers

- Morgan
- JoJo
- Ieva

Fetch Followers

- /user/:id
- /user/:id/posts
- **/user/:id/followers**

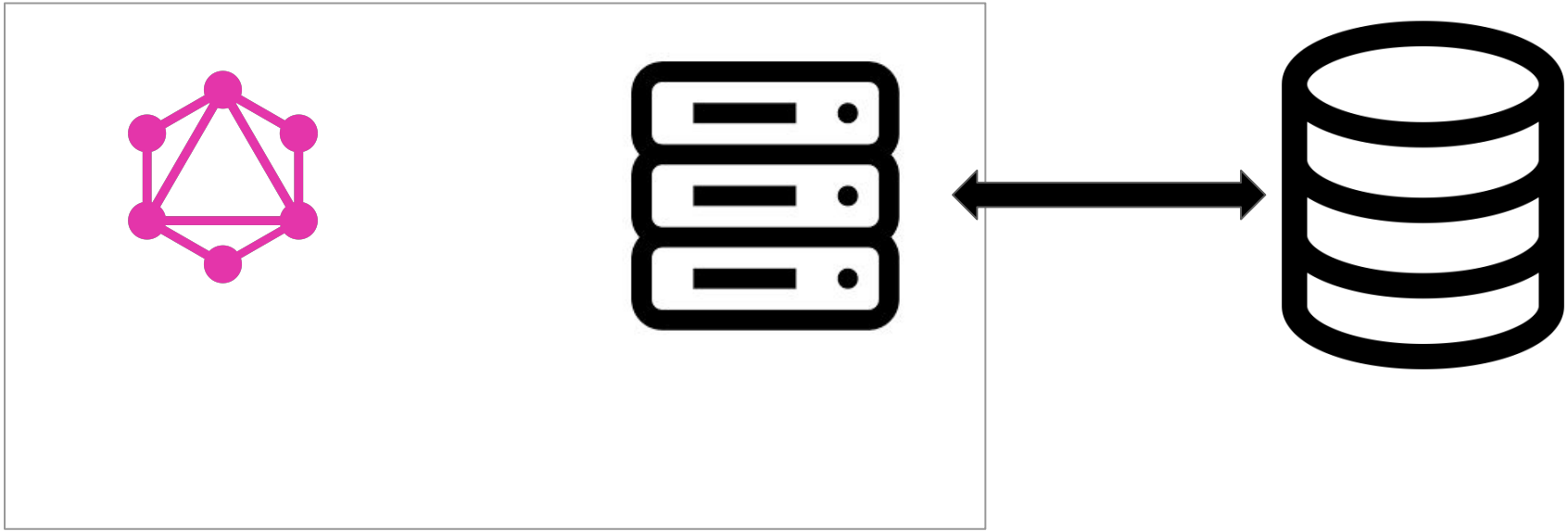


HTTP GET

```
{
    "Followers": [{
        "Id": "ebg14frhkfhd"
        "Name": "Morgan"
        "Address": "Somewhere else"
        "Birthday": "15.06.1988"
        ...

    },
    {JoJo}, {Ieva}, ...]
}
```

Blogging App in GraphQL



Blogging App in GraphQL

MoHo

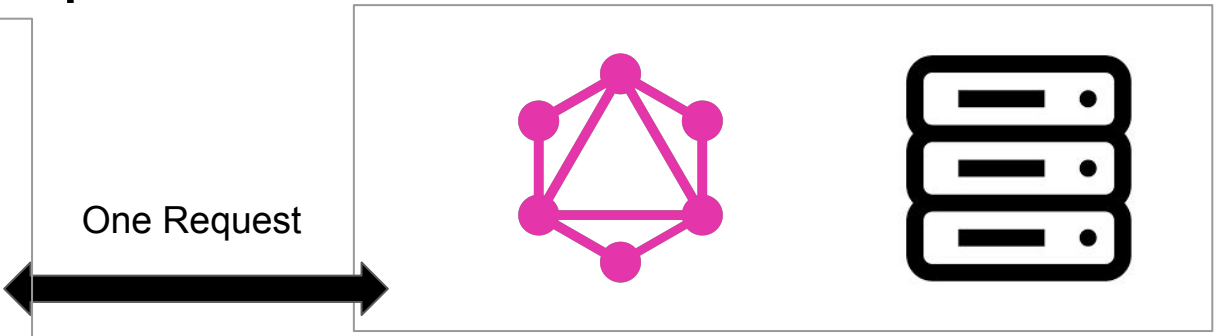
MoHo's posts:

- Learn about GraphQL today
- Do you React?
- What's Next.js?
- TypeScript could help maybe if...

Last 3 Followers

- Morgan
- JoJo
- Ieva

One Request



HTTP POST

```
Query{
      User(id: "ebg14rh7kfhd"){
            Name
            Posts{
                  Title
            }
            followers(last: 3){
                  Name
            }
      }
}
```

Blogging App in GraphQL

MoHo

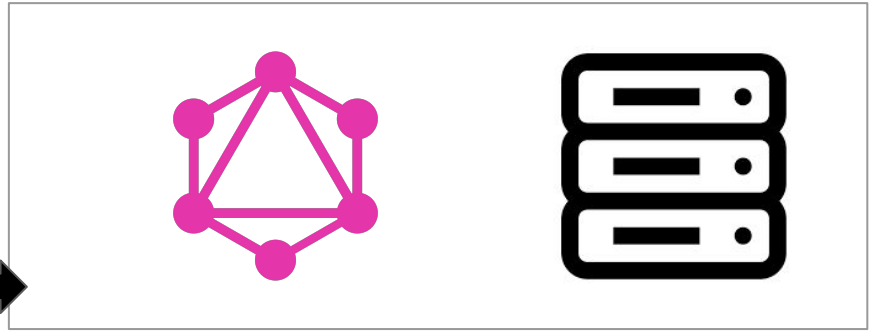
MoHo's posts:

- Learn about GraphQL today
- Do you React?
- What's Next.js?
- TypeScript could help maybe if...

Last 3 Followers

- Morgan
- JoJo
- Ieva

One Response




```
{
    "Data": {
        "User":{
            "Name": "MoHo",
            "Posts": [
                { "title": "Learn about GraphQL today"},
                { "title": "Do you React?"},
                { "title": "What's Next.js?"},
                { "title": "TypeScript could help maybe if..."}
            ],
            "Followers": [
                { "name" : "Morgan"},
                { "name" : "JoJo"},
                { "name" : "Ieva"}
            ]
```

Request vs Response in GraphQL

Request Query

```
Query{
     User(id: "ebg14rh7kfhd"){
          Name
          Posts{
               Title
          }
          followers(last: 3){
               Name
          }
     }
}
```

Response

```
{
     "Data": {
          "User":{
               "Name": "MoHo",
               "Posts": [
                  { "title": "Learn about GraphQL today"},
                  { "title": "Do you React?"},
                  { "title": "What's Next.js?"},
                  { "title": "TypeScript could help maybe if..."}
               ],
               "Followers": [
                    { "name" : "Morgan"},
                    { "name" : "JoJo"},
                    { "name" : "Ieva"}
               ]
          }
     }
}
```

A more efficient alternative to REST

- Increased mobile usage creates need for efficient data loading
- Fast development and expectation for rapid feature development - Versioning
- Nullability
- Pagination
- Server-side Batching

GraphQL vs REST

- Scalability
  - With **REST** you may need to call multiple call to assemble a complete view
- Performance
  - Smaller payload sizes - You ask for what you need. Not all provided data
- **GraphQL**: The ability to batch requests, where you can define dependencies between two separate queries and fetch data efficiently.
- **GraphQL**: The ability to create subscriptions, where your client can receive new data when it becomes available.
- **GraphQL** is always the smallest possible request.
- **REST** generally defaults to the fullest. It's common practice to offer options like `?fields=foo,bar`

GraphQL vs REST

- **GraphQL**: predictable responses
- **REST**: Deciding on URI schema get tough when we start to have heavily nested relationship
- **REST**: Sometimes too many HTTP request
- **GraphQL**: Generate documentation
- **REST**: Caching easier

Use cases

Architectural Use cases

1. GraphQL server with a connected database

- Common for *greenfield* projects
- Uses single web server that implements GraphQL
- Server resolve queries and constructed response with data that it fetches from DB
- GraphQL doesn't care about DB or format of stored data. (SQL, NoSQL)

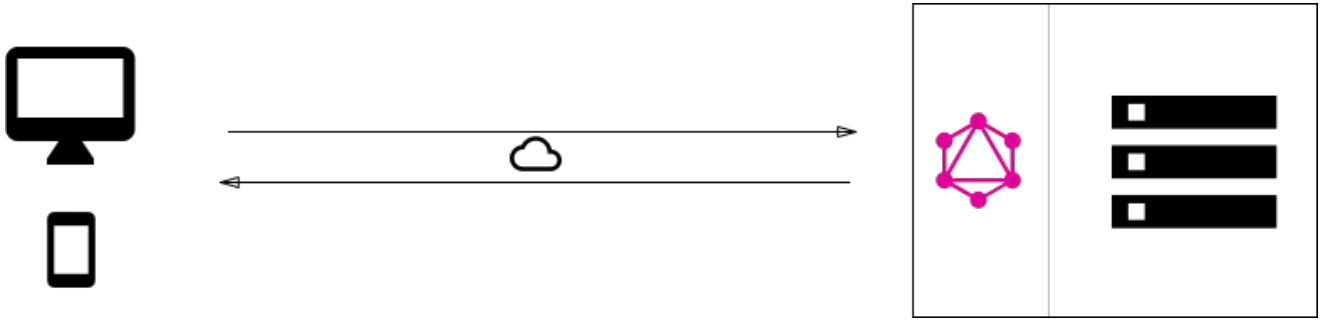


Image from http://howtographql.com

Architectural Use cases

2. GraphQL layer that integrates existing systems

- Companies with legacy products and many different APIs
- GraphQL can unify existing systems and hide complexity of data fetching logic
- GraphQL doesn't care about data sources(DB, another api, web services)

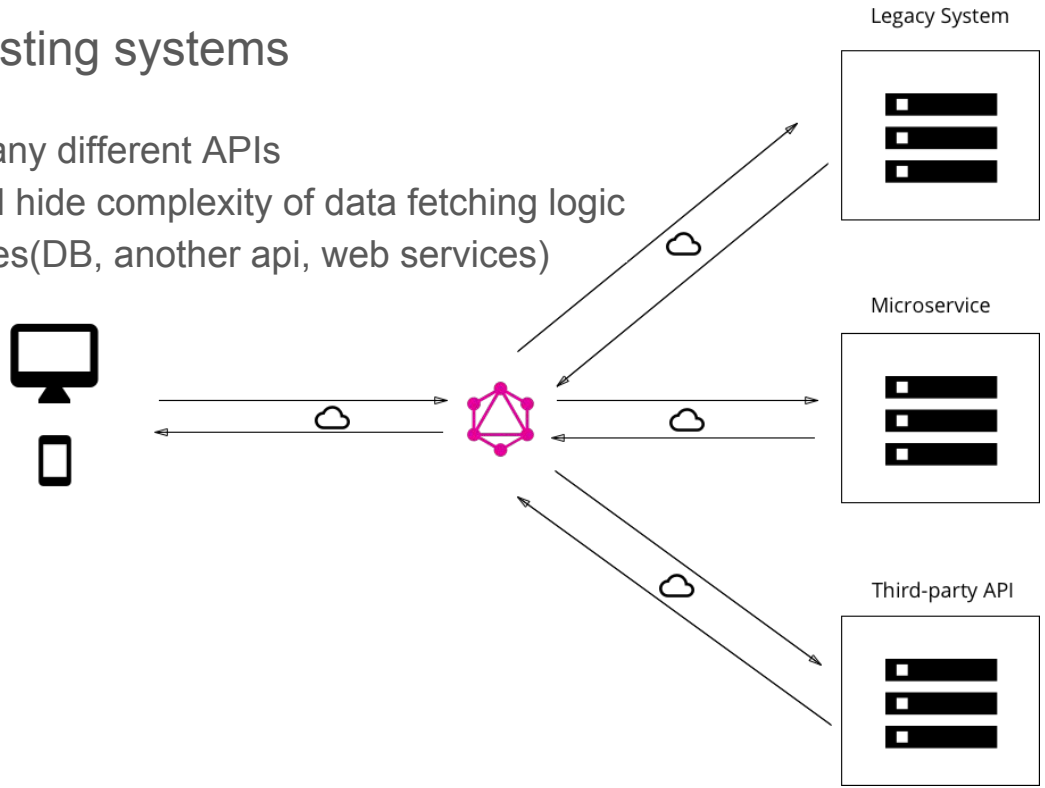


Image from http://howtographql.com

Architectural Use cases

3. Hybrid approach with connected DB and integrated systems

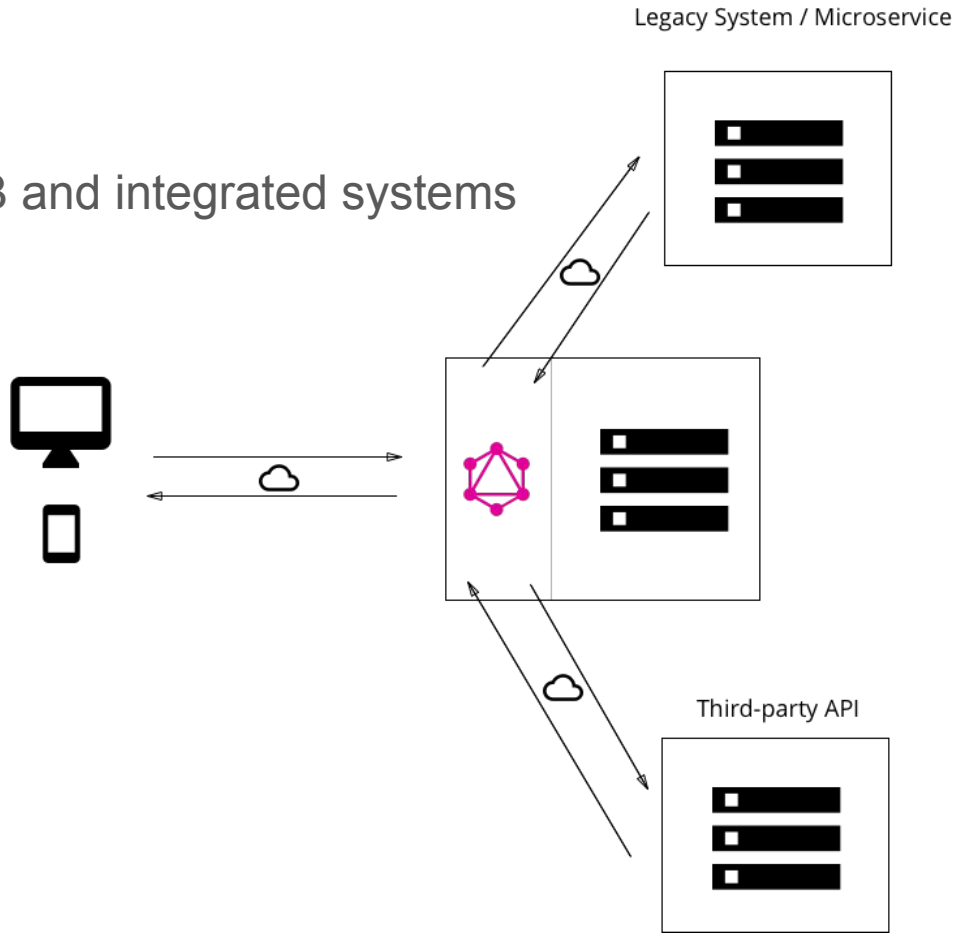


Image from http://howtographql.com

Resolvers function

```
query{
    user(id: "123"){
        name
        friends(first: 3){
            name
            age
        }
    }
}
```

Resolver

```
User(id: String!) User

name(user: User!) String
age(user: User!) Int

friends(first: Int, user: User!):
[User!]!
```

Live demo

DEMO TRADING

Brokers want you to think its real

Questions...

Error Handling

```
{
  "data": { ... },
  "errors": [ ... ]
}
```

Authentication and Authorization

- Authentication with OAuth
- Authorization in business logic layer

Server-side caching

It is not clear what a client will request next, so putting caching layer right behind the API doesn’t make a lot of sense.

Server-side caching still is a challenge with GraphQL.

Thank you