

GraphQL

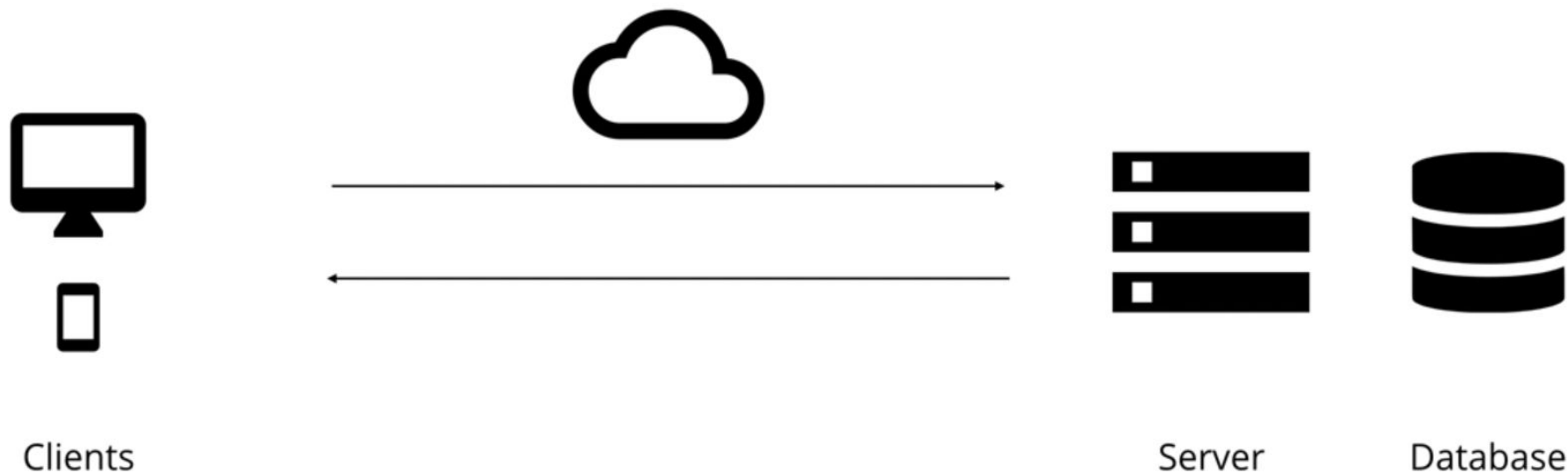
What is GraphQL?

GraphQL is a query language for your API, It was developed and [open-sourced by Facebook](#).

It enables declarative data fetching.

GraphQL server exposes single endpoint and responds to queries.

A Query Language for APIs






A more efficient alternative to REST

- ❖ Increased mobile usage creates need for efficient data loading.
- ❖ Variety of different frontend frameworks and platforms on the client side.
- ❖ Fast development speed and expectation for rapid feature development.

Why GraphQL is better than REST

Example: Blogging App with REST

3 API endpoints

<code>/users/<id></code>	
<code>/users/<id>/posts</code>	
<code>/users/<id>/followers</code>	

Mary's posts:

Last three followers:



HTTP GET

```
{  
  "user": {  
    "id": "er3tg439frjw"  
    "name": "Mary",  
    "address": { ... },  
    "birthday": "July 26, 1982"  
  }  
}
```

/users/<id>

/users/<id>/posts

/users/<id>/followers



Mary

Mary's posts:

Learn GraphQL Today

React & GraphQL - A declarative love story

Why GraphQL is better than REST

Relay vs Apollo - GraphQL clients

Last three followers:

John, Alice, Sarah



HTTP GET

```
{  
  "followers": [{  
    "id": "leo83h2dojsu"  
    "name": "John",  
    "address": { ... },  
    "birthday": "January 6, 1970"  
  }, {  
    "id": "die5odnvlsl0"  
    "name": "Alice",  
    "address": { ... },  
    "birthday": "May 1, 1989"  
  }]  
}
```

/users/<id>

/users/<id>/posts

/users/<id>/followers



Advantages of GraphQL

Overfetching : downloading unnecessary data

Underfetching: An endpoint does not return enough of the right information; need to send multiple requests.

Benefit of schemas and types

GraphQL uses strong type system to define capabilities of an API.

Schema serves as contract between client and server.

Frontend and backend teams can work completely independent from each other.

The Schema Definition Language (SDL)

Adding a relation

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

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

Person



Post

Fetching Data with Queries

```
{  
  allPersons {  
    name  
    posts {  
      title  
    }  
  }  
}
```

```
{  
  "allPersons": [  
    {  
      "name": "Johnny",  
      "posts": [  
        { "title": "GraphQL is awesome"},  
        { "title": "Relay is a powerful GraphQL Client"}  
      ]  
    },  
    {  
      "name": "Sarah",  
      "posts": [  
        { "title": "How to get started with React & GraphQL" }  
      ]  
    },  
    {  
      "name": "Alice",  
      "posts": []  
    },  
  ]  
}
```

Writing data with mutations?

3 kinds of mutations:

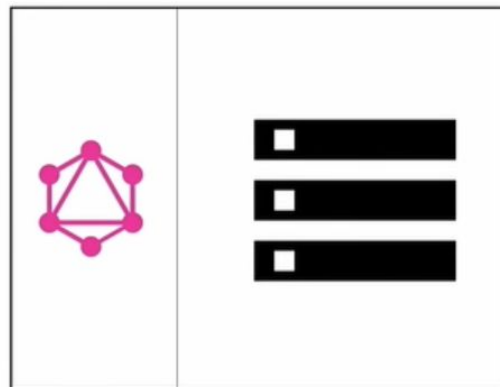
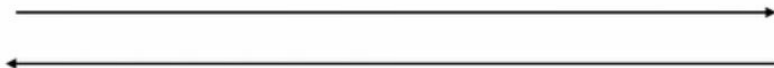
Creating new data

Updating existing data

Deleting existing data

Writing Data with Mutations

```
mutation {  
  createPerson(name: "Bob", age: 36) {  
    name  
    age  
  }  
}
```

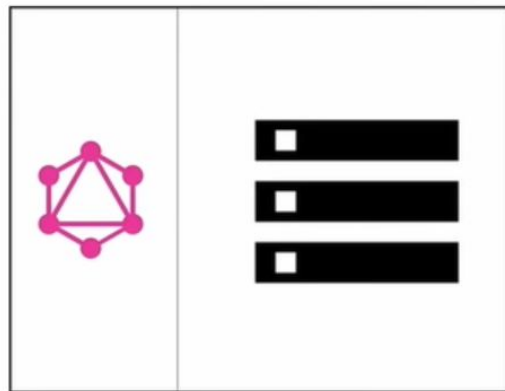


Taking real time connection with server

Realtime Updates with Subscriptions



```
subscription {  
  newPerson {  
    name  
    age  
  }  
}
```



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

```
type Mutation {  
  createPerson(name: String!, age: String!): Person!  
  updatePerson(id: ID!, name: String!, age: String!): Person!  
  deletePerson(id: ID!): Person!  
  createPost(title: String!): Post!  
  updatePost(id: ID!, title: String!): Post!  
  deletePost(id: ID!): Post!  
}
```

```
type Subscription {  
  newPerson: Person!  
  updatedPerson: Person!  
  deletedPerson: Person!  
  newPost: Post!  
  updatedPost: Post!  
  deletedPost: Post!  
}
```

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

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

GraphQL is actually transport layer ignostic (It can use with any available network protocol like TCP/WebSocket)

GraphQL does not care about the database or the format that is used to store the data.

Some of graphql architectural use cases

GraphQL server with a connected database.

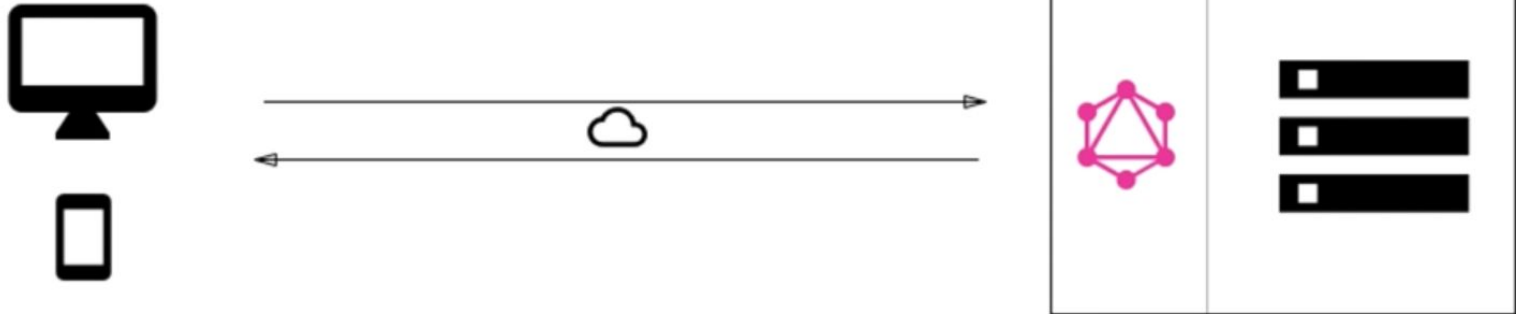
GraphQL server to integrate existing system.

A hybrid approach with a connected database and integration of existing system.

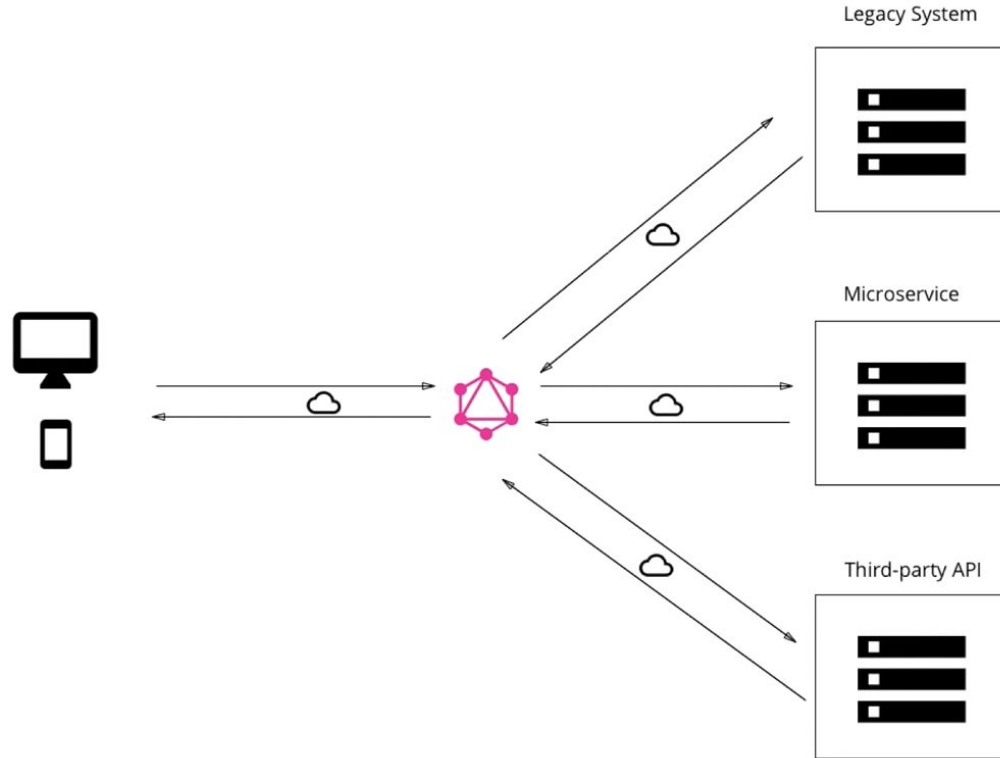
Use Case 1: GraphQL server with a connected database

Uses single web server that implements GraphQL.

Server resolves queries and constructs response with data that it fetches from the database.



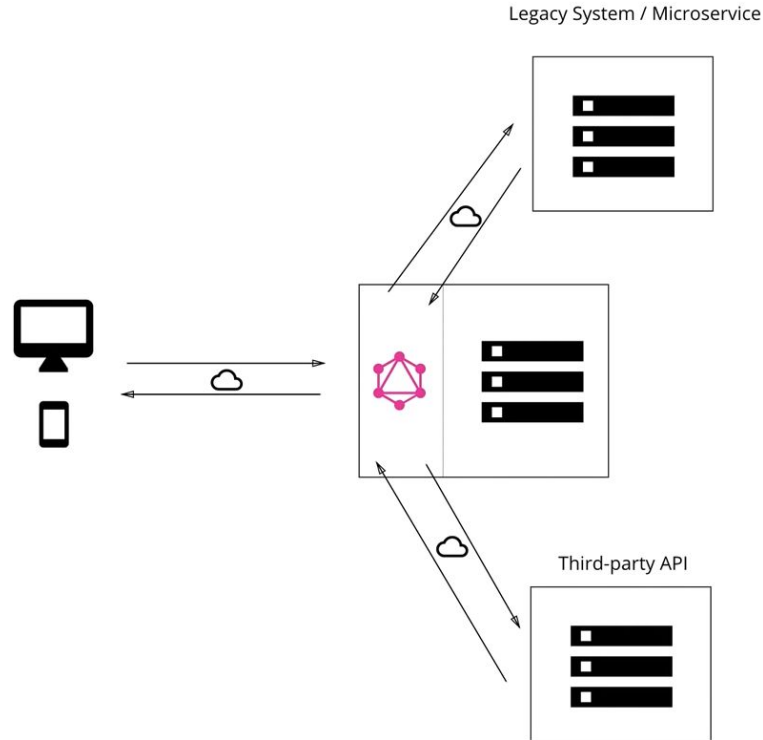
Use Case 2: GraphQL server integrating existing system



Compelling use case for companies with legacy infrastructures and many different APIs.

GraphQL can be used to unify existing systems and hide complexity of data fetching logic.

Use Case 3 : Hybrid approach with connected database & integrated systems



How GraphQL able to cope with all different environments ?
in the concept Resolver functions

Resolver functions

GraphQL queries/mutations consist of set of fields

GraphQL server has one resolver function per field.

The purpose of each resolver is to retrieve the data for its corresponding field

Resolver functions

```
query {  
  User(id: "er3txsa9frju") {  
    name  
    friends(first: 5) {  
      name  
      age  
    }  
  }  
}
```



Resolvers



```
User(id: String!): User  
name(user: User!): String!  
age(user: User!): Int!  
friends(first: Int, user: User!): [User!]!
```