**What is GraphQL –**

* GraphQL is query language for the API and provides server side runtime for executing queries. These queries are executed against type system defined for the data.
* At its core GraphQL enables declarative data fetching. Clients can specify exactly what data it needs from API.
* Instead of multiple end points that return fixed data structure, GraphQL server exposes single endpoint and responds with precisely data client asks for.
* It was invented and opensourced by facbook and now maintained by large community of companies and individuals.
* GraphQL can be implemented in many languages – c#/ .NET, Clojure, Elixir, Erlang, Go, Groovy, Java, Javascript,, Julia, Kotlin, Perl, PHP, Python, R, Ruby, Rust, Scala, Swift. Example implementation for each is here ->

<https://graphql.org/code/>

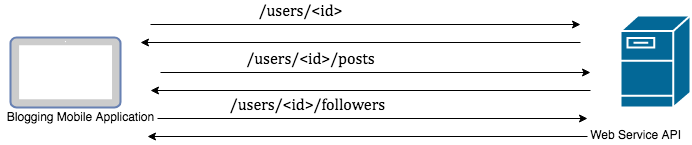
* Simplest of implementation GraphQL can be seen as middle layer between client and server. It starts making sense when multiple clients are communicating with multiple datasources. (webservices/ databases).

**REST vs GraphQL**

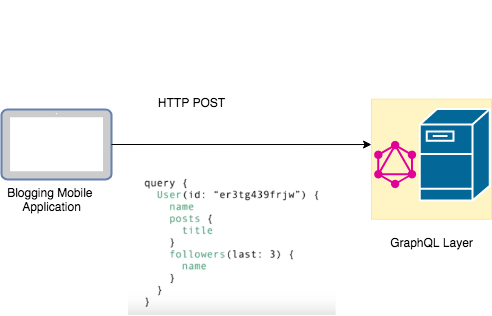
Let us take example of blogging application. Following is the structure –

* There are 3 endpoints ->
  + /users/<id> to fetch specific user
  + /users/<id>/posts to fetch posts specific to user
  + /users/<id>/followers to fetch followers for user.
* Mobile application has to display posts specific to user and followers

Following will be the flow of API calls with REST approach –



Each of the call might return more data than needed by front end. Also 3 different calls mean we are consuming more network resources from device.



With GraphQL layer there is only one POST request hence avoiding multiple requests. Also schema defines exactly what fields are required by client.

**GraphQL Core Concepts -**

* **Schema Definition language –**
  + GraphQL has its own type system to define schema of API. The syntax for writing schema is called Schema Definition Language (SDL).
* **GraphQL Types**
  + We can SDL to define simple and complex types e.g.

IN following example User is associated with Post type

Type User {

name : String!

age : Int!

}

type Post {

title: String!

author: User!

}

* **GraphQL Schema**

GraphQL defines the contract between client and server.

It is collection of GraphQL types with special root types (query, mutation, subscription)

* **Query, Mutation & Subscription**
  + In GraphQL fetching data is achieved with root type - Query. E.g.

query {

users {

Name

Post {

title

}

}

}

Here we ask API for all users with name field and titles of all their posts.

* + Writing of Data to backend is achieved with Mutations (Creating/ Delete/ modify).

mutation {

createUser(name : “Bob” , age: 35) {

name

age

}

}

above query issues create request to backend. If the request is success this returns name and age of created user.

Another requirement for applications is to get realtime notification about the changes to backend systems. This is handled by GraphQL subscriptions. E.g.

Subscription {

newUser {

name

age

}

}

As and when the new User is created server will push new data to client over websocket connection.

**Resolver Functions**

Resolver functios are responsible for retrieving data for its corresponding set of fields.

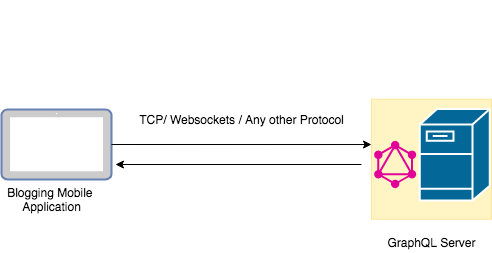
e.g. the resolver function to fetch user details will look something like ->

User(id : String!) : User

**GraphQL Architecture Use cases –**

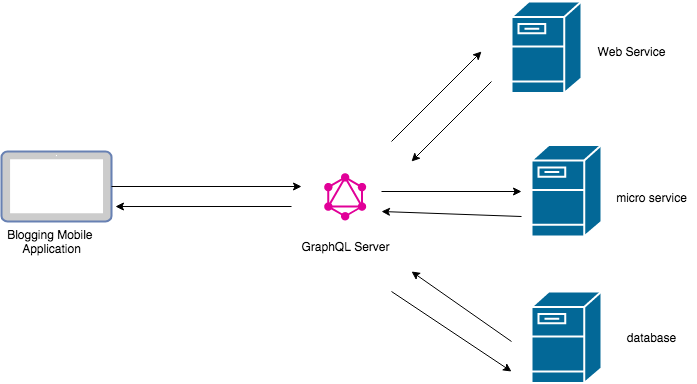
1. **GeaphQL server with connected database ->**

* **use case for newly implemented projects**
* **single web server that implements GraphQL.**
* **Server resolves queries from front end and constructs response and returns to client.**
* **GraphQL is protocol agnostic. That means communication can be on TCP or websockets or any other protocol.**
* **It can connect to any database (sql db or nosql db)**

****

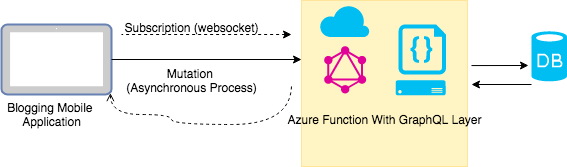
**2. GraphQL Server integrating with existing system**

* **GraphQL can be used to unify all existing systems and hide complexity of data fetching logic.**
* **Existing system can be web service or database.**

****

1. **Realtime serverless API**

* **GraphQL API can be deployed as azure functions.**
* **Front end application will initiate mutation by invoking API which create event which results in Azure function invocation.**
* **Front end application has subscription with GraphQL layer which works over web socket connection.**
* **Once azure function finishes task it raises event which result in notification sent to front end application as it has subscription open with GraphQL API.**

****