WHAT IS GRAPHQL?

GraphQL is an open-source data query and manipulation language for APIs and a runtime for fulfilling queries with existing data.

GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data. With GraphQL, you gain a complete and understandable description of the data in your API. Additionally, it provides clients the power to ask for exactly what they need and nothing more through its streamlined interface. A considerable benefit of GraphQL is that it makes it easier to evolve APIs over time and enables powerful developer tools."

Some of the developer tools and languages that GraphQL servers integrate with are Haskell, JavaScript, Perl, Python, Ruby, Java, C++, C#, Scala, Go, Rust, Elixir, Erlang, PHP, R, D, and Clojure.

GraphQL allows making multiple resources requests in a single query call, which saves a lot of time and bandwidth by reducing network round trips to the server. With clients being able to ask for exactly what they need and nothing more, there is no need to filter through the data response.

GraphQL vs. REST API

Even though REST API is a standard API tool, there are significant advantages to using GraphQL as an alternative. For starters, with REST API, multiple endpoints exist for addressing different entities, but related entities have to be queried independently. GraphQL allows querying for related entities from one entity.

REST requires that an endpoint be defined before data can be queried for entries, forcing a developer to coordinate each use case ahead of time. A developer needs to know precisely how each request will work before they begin each stage of their project.

Since REST API calls frequently fetch too much data from an endpoint, it can become difficult to ensure that the front end receives the correct data set(s). GraphQL solves this problem because the same data set is declared, removing any doubt that the accurate information is present.

Low-powered devices became more common. Slow network connections and the increased need for fast responses led to fewer requests to fetch all required data. Developers needed an alternative to REST due to the changing digital landscape.

Building Schema in GraphQL

A GraphQL schema is at the core of any GraphQL server implementation. It describes the functionality available to the client applications that connect to it. We can use any programming language to create a GraphQL schema and build an interface around it.

The GraphQL runtime defines a generic graph-based schema to publish the capabilities of the data service it represents. Client applications can query the schema within its capabilities. This approach decouples clients from servers and allows both to evolve and scale independently.

In GraphQL, if a field does not have a resolve function, GraphQL will examine the parent value returned by the parent field's resolver. Under the assumption it is an Object, it will try to find a property on that parent value that matches the field name.

If it finds a match, it resolves the field to that value or call the function corresponding to the property and then resolves to the value returned by the function.

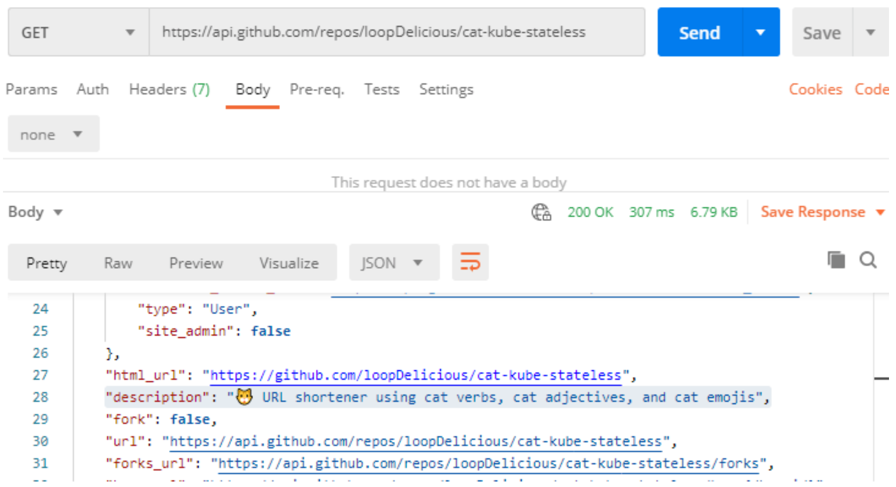
In the example below, the hello field has no resolver, but it has a root value that serves as the parent field, and that has a field called hello which is a function. So GraphQL calls the function and then resolves to the value returned by the function.

```
==========GraphQLSchema

const { graphql, GraphQLSchema, GraphQLObjectType, GraphQLString } =
require('graphql');

const schema = new GraphQLSchema({
  query: new GraphQLObjectType({
    name: 'Query',
    fields: () => ({
      hello: {
        type: GraphQLString,
        resolve: () => 'Hello world!'
      }
    })
  })
});

graphql(schema, '{ hello }').then((response) => {
  console.log(response);
});
```

The Benefits of GraphQL

Some standard implementations of GraphQL provide only a single endpoint for the full implementation and specify the entry entity in the provided query.

With more devices and frameworks, developers could now build web applications faster than ever before. Unfortunately, they were frequently slowed down by REST's lack of flexibility for changing API endpoints.



GET https://api.github.com/repos/loopDelicious/cat-kube-stateless

none

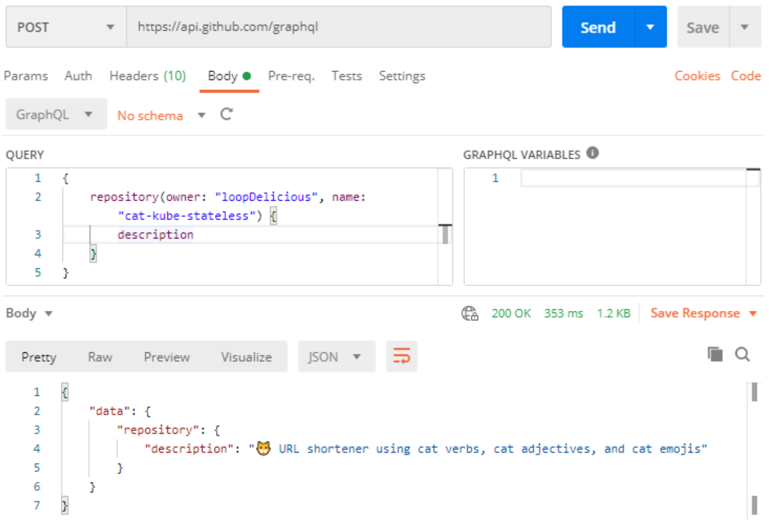
This request does not have a body

Body 200 OK 307 ms 6.79 KB

JSON

```
        "type": "User",
        "site_admin": false
    },
    "html_url": "https://github.com/loopDelicious/cat-kube-stateless",
    "description": "🐱 URL shortener using cat verbs, cat adjectives, and cat emojis",
    "fork": false,
    "url": "https://api.github.com/repos/loopDelicious/cat-kube-stateless",
    "forks_url": "https://api.github.com/repos/loopDelicious/cat-kube-stateless/forks",
```

i.



POST https://api.github.com/graphql

GraphQL No schema

QUERY

```
{
    repository(owner: "loopDelicious", name:
        "cat-kube-stateless") {
        description
    }
}
```

GRAPHQL VARIABLES

Body 200 OK 353 ms 1.2 KB

JSON

```
{
    "data": {
        "repository": {
            "description": "🐱 URL shortener using cat verbs, cat adjectives, and cat emojis"
        }
    }
}
```

ii.

GraphQL is perfect for larger applications (such as the Facebook Newsfeed that sparked its creation).

There's less to worry about when it comes to fetching data, and sometimes the results can be achieved much quicker.

GraphQL decouples apps from services so that it is possible for app developers to describe the data they need. GraphQL then fetches the data requested by the query from the data sources defined in the service.

GraphQL queries are not faster than REST queries, but because you can pick the fields you want to query, GraphQL requests will always be smaller and more efficient. GraphQL is unlike REST, where additional data is often returned, even when that data isn't necessary.

Overall, GraphQL provides a complete and understandable description of the data in your API and gives clients the power to ask for exactly what they need and nothing more.