Google

Intro to GraphQL

Nat Welch / SRECon'19 Americas

2019-03-26

Nat Welch
@icco

Nat Welch is an SRE based in Brooklyn, NY, and the author of "Real World SRE".

He currently works for Google on the Customer Reliability Engineering team.

In the past, he has worked for First Look Media, Hillary for America, iFixit, and others.

Agenda

- What is GraphQL
- Why use GraphQL
- Using GraphQL in production
- Current state of the ecosystem

What is GraphQL?

> GraphQL is a query language designed to build client applications by providing [...] a system for describing their data requirements and interactions.

GraphQL June 2018 Spec
Facebook

Source: https://facebook.github.io/graphql/June2018/#sec-Overview

GraphQL is a specification which defines a schema on an API server, which validates client calls.

GraphQL is an application layer. GraphQL does not define how data is stored or queried from the source.

```
$ curl -d '' \
    https://graphql.natwelch.com/graphql
```

The JSON response to GraphQL queries mirror each other. This can be nested as deep as the client wants.

The shape of the request defines the shape of the response.

```
query {
  time
}
```

```
{
  "data": {
    "time": "2019-03-24T21:32:22Z"
  }
}
```

```
query {
  post(id: "691") {
    title
  }
}
```

```
{
  "data": {
    "post": {
      "title": "What's making me happy"
    }
  }
}
```

GraphQL is strongly typed. The schema defines types and object relationships.

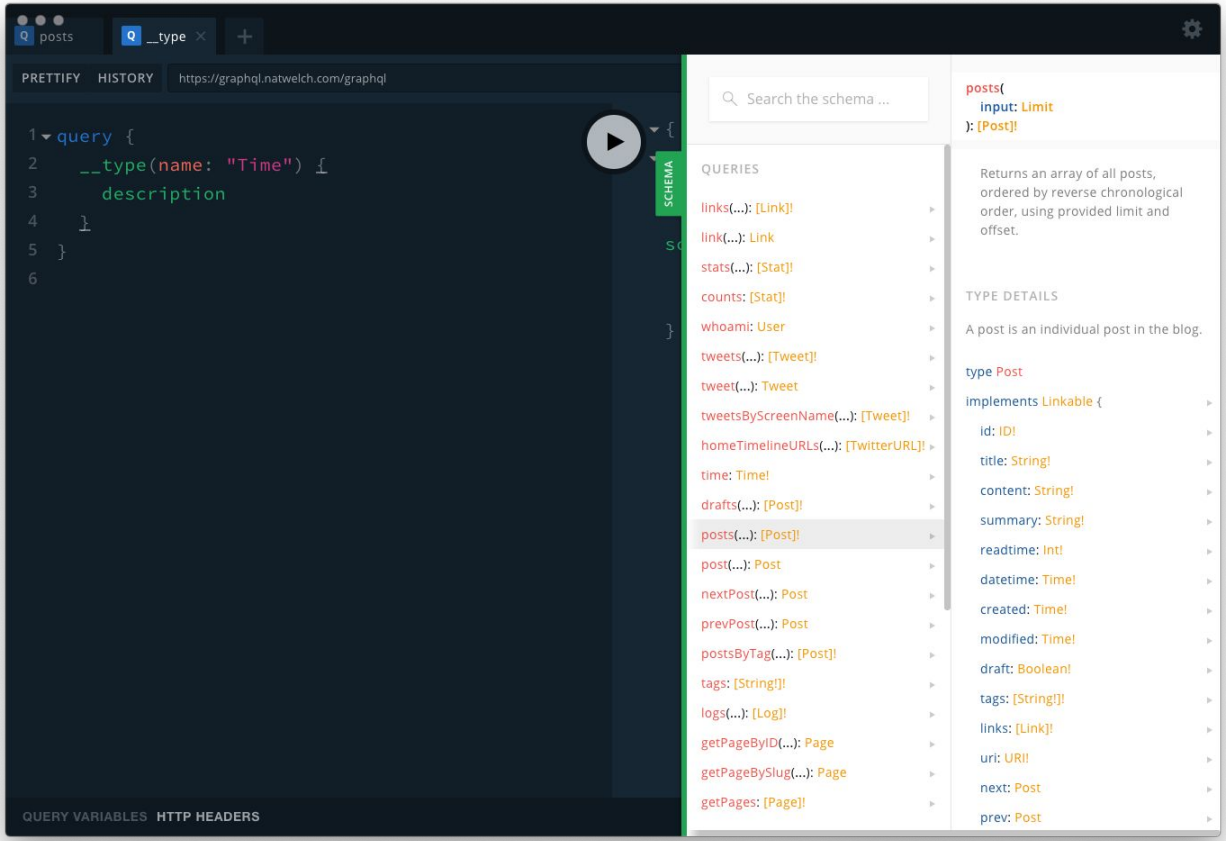
```
query {
  post(id: "691") {
    related(input: { limit: 7 }) {
      id
    }
  }
}
```

```
{
  "data": {
    "post": {
      "related": [
        {
          "id": "457"
        },
        {
          "id": "663"
        },
```

GraphQL has documentation as a first class citizen. Clients can ask servers for information about schema, including documentation and types.

```
query {
  __type(name: "Time") {
    description
  }
}
```

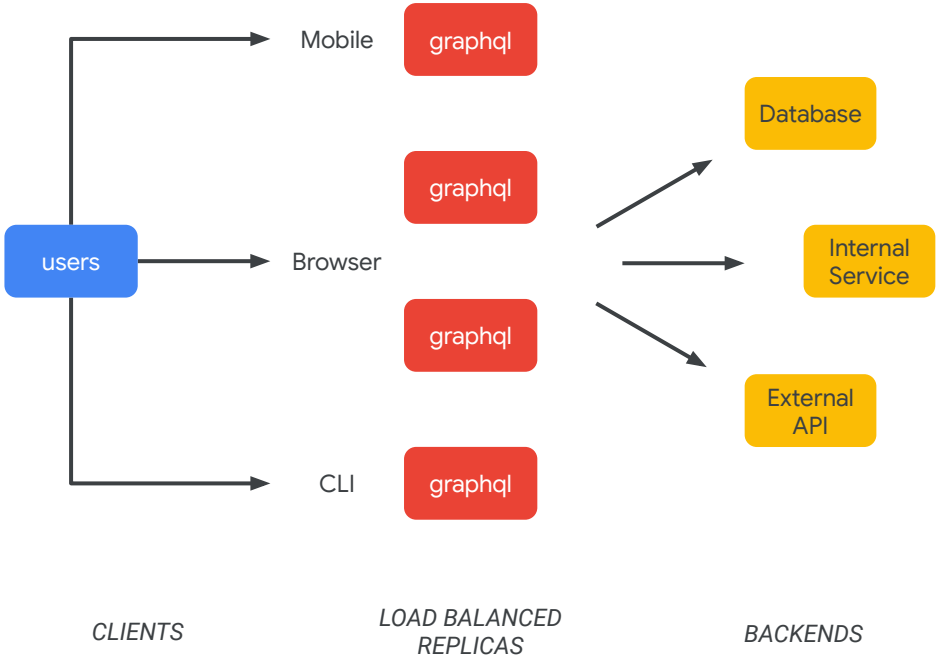
```
{
  "data": {
    "__type": {
      "description": "Time is a datetime
scalar with timezone."
    }
  }
}
```



Source: https://github.com/prisma/graphql-playground

Why

- Separate concerns between frontend and backend
- Lower dependencies of client launches
  - Client is blocked on backend launch and vice-versa less frequently
- Unify servers that are accessible to the public
  - Proxy to other backend services
  - One API, many data sources



Use Cases

- Media Companies
  - New York Times[1], First Look Media[2], Major League Soccer[3]
- Mobile Apps
  - Artsy[4], Yelp[5]
- Complex Datasets
  - Github[6], Shopify[7], Fabric[8]

1. https://open.nytimes.com/react-relay-and-graphql-under-the-hood-of-the-times-website-redesign-22fb62ea9764
2. https://code.firstlook.media/welcome
3. https://labs.mlssoccer.com/implementing-graphql-at-major-league-soccer-ff0a002b20ca
4. http://artsy.github.io/blog/2016/11/02/improving-page-speed-with-graphql/
5. https://engineeringblog.yelp.com/2017/05/introducing-yelps-local-graph.html
6. https://githubengineering.com/the-github-graphql-api/
7. https://help.shopify.com/en/api/custom-storefronts/storefront-api/graphql
8. https://fabric.io/blog/building-fabric-mission-control-with-graphql-and-relay

Production Problems

Dumb Resolvers

- Really easy to make resolvers that make a lot of queries.
- Use some sort of project like dataloader[1] that minimizes or unifies queries to backends.

1. https://github.com/facebook/dataloader

Fan Out

- Similar to dumb resolvers, it is really easy to write a system that turns one graphql request into thirty backend requests.
- One metric to monitor is backend requests per query. If that spikes, or goes out, you may need to refactor your schema.

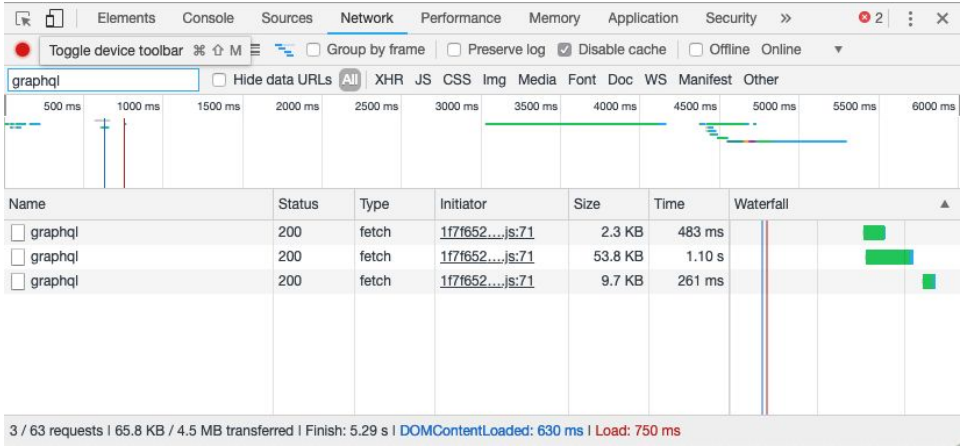
Query Complexity

It is really easy to make complex requests. If you can, put hard limits on query complexity. Otherwise you could find people sending abusive queries.

```
query {
  posts({limit: 1000}) {
    related {
      title
      uri
      related {
        title
        uri
        related {
          ...
        }
      }
    }
```

Request Size

- Really easy to make gigantic requests
- Also easy to make gigantic responses



Caching

- Doable, just not on by default, different-ish from traditional REST endpoints
- Most GraphQL endpoints use POST which is not cached by CDNs by default
- Persisted queries
  - Apollo Automatic Persistent Queries (on-the-fly)[1]
  - Relay v3 Persisted Queries (pre-compiled)[2]
  - Support varies by server
- Also can cache in client code, not always relying on cache headers

1. https://github.com/apollographql/apollo-link-persisted-queries
2. https://facebook.github.io/relay/docs/en/persisted-queries.html

Ecosystem

Different Directions, Same Goal

Schema First

Group of frameworks focused on generating code based off of schema or requiring schema to be written.

Code First

Group of frameworks focusing on defining backends and resolvers and auto-generating schema from that.

Some interesting things to look into

- Apollo - Hosted GraphQL plus popular JS framework
- gqlgen - Go servers generated from schema
- Prisma & Hasura - Pushing GraphQL closer to the database
- Subscriptions
- Fragments

Thanks!

DM @icco or come find me in-person if you have questions.

Google