



Intro to GraphQL

A solid blue horizontal bar is positioned directly beneath the first letter of the title "Intro to GraphQL".

Nat Welch / SRECon'19 Americas

2019-03-26

nat.wiki/srecon-graphql

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

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"  
    }  
  }  
}
```

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

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


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

```
type Query {  
  time: Time  
  post(id: ID): Post  
}
```

```
type Post {  
  title: String  
  related(input: Limit): [Post]  
}
```

```
"""
```

```
Time is a datetime scalar with timezone.
```

```
"""
```

```
scalar Time
```

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."
    }
  }
}
```

```
"""
```

```
A post is an individual post in the blog.
```

```
"""
```

```
type Post implements Linkable {  
  title: String!
```

```
  "A list of related posts."
```

```
  related(input: Limit!): [Post]!
```

```
}
```


posts

__type

PRETTIFY

HISTORY

https://graphql.natwelch.com/graphql

```
1 query {
2   __type(name: "Time") {
3     description
4   }
5 }
6
```

SCHEMA

Search the schema ...

QUERIES

links(...): [Link!] ▶

link(...): Link ▶

stats(...): [Stat!] ▶

counts: [Stat!] ▶

whoami: User ▶

tweets(...): [Tweet!] ▶

tweet(...): Tweet ▶

tweetsByScreenName(...): [Tweet!] ▶

homeTimelineURLs(...): [TwitterURL!] ▶

time: Time! ▶

drafts(...): [Post!] ▶

posts(...): [Post!] ▶

post(...): Post ▶

nextPost(...): Post ▶

prevPost(...): Post ▶

postsByTag(...): [Post!] ▶

tags: [String!]! ▶

logs(...): [Log!] ▶

getPageByID(...): Page ▶

getPageBySlug(...): Page ▶

getPages: [Page!] ▶

posts(

Input: Limit

): [Post!]

Returns an array of all posts, ordered by reverse chronological order, using provided limit and offset.

TYPE DETAILS

A post is an individual post in the blog.

type Post

implements Linkable {

id: ID! ▶

title: String! ▶

content: String! ▶

summary: String! ▶

readtime: Int! ▶

datetime: Time! ▶

created: Time! ▶

modified: Time! ▶

draft: Boolean! ▶

tags: [String!]! ▶

links: [Link!] ▶

uri: URI! ▶

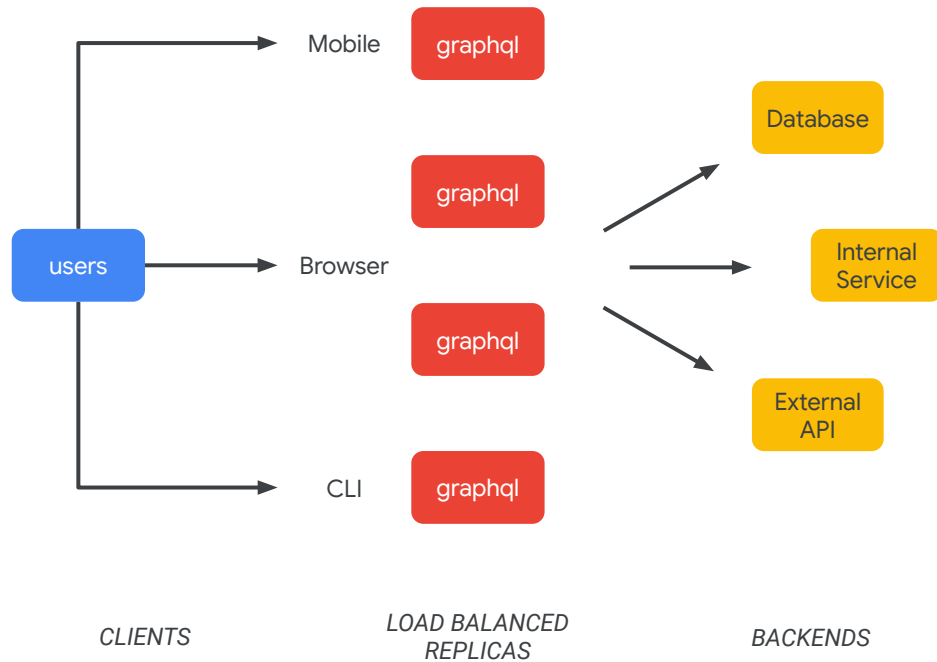
next: Post ▶

prev: Post ▶

QUERY VARIABLES

HTTP HEADERS

Why?



Why

- Simplify client development
- Simplify backend versioning
- Separate concerns between frontend and backend
- Lower dependencies of client launches
- Single front-end that is accessible to the public

Use Cases

- Media Companies
 - New York Times¹, First Look Media², Major League Soccer³
- Mobile Apps
 - Artsy⁴, Yelp⁵
- Complex Datasets
 - Github⁶, Shopify⁷, Fabric⁸

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 dataloader¹ 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

Name	Status	Type	Initiator	Size	Time	Waterfall	▲
<input type="checkbox"/> graphql	200	fetch	1f7f652....js:71	2.3 KB	483 ms		
<input type="checkbox"/> graphql	200	fetch	1f7f652....js:71	53.8 KB	1.10 s		
<input type="checkbox"/> graphql	200	fetch	1f7f652....js:71	9.7 KB	261 ms		

Caching

- 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
- 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
- Unions & Interfaces

Thanks!

nat.wiki/srecon-graphql



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