

# GraphQL from the Ground Up

Jason Clark

New Relic, Principal Engineer and Architect

With special thanks to Irene Lopez



@jasonrclark@ruby.social



jasonrclark

# Where We're Going Today

- What's GraphQL?
- Why choose GraphQL?
- How to build it?
- How did it go?



@jasonrclark@ruby.social



jasonrclark

# What's GraphQL?

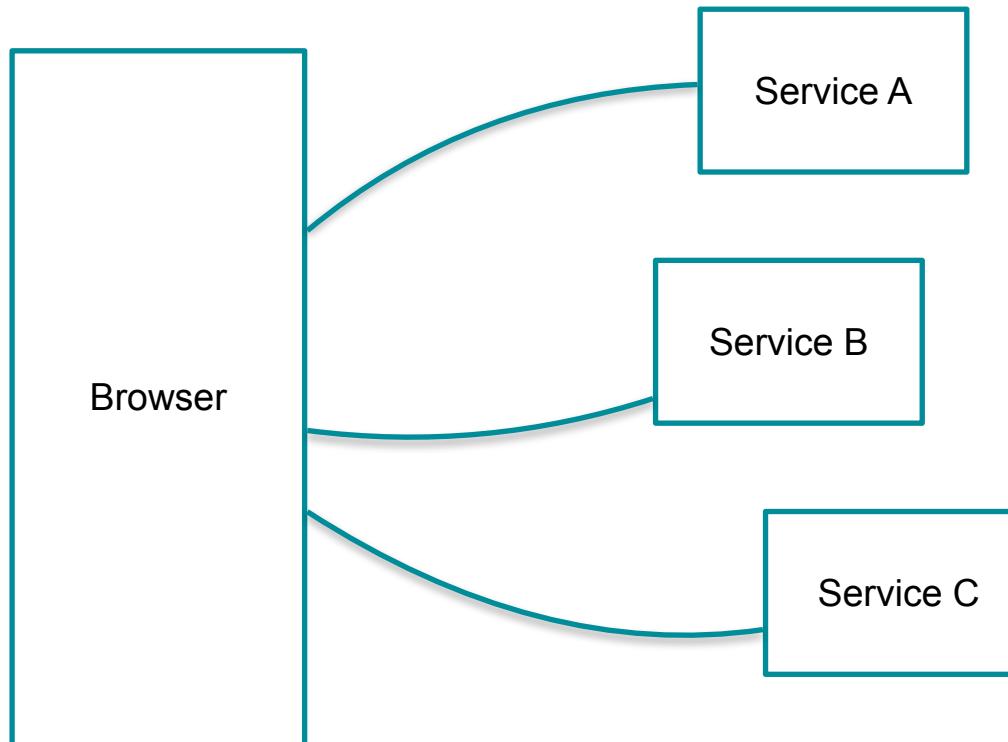


@jasonrclark@ruby.social

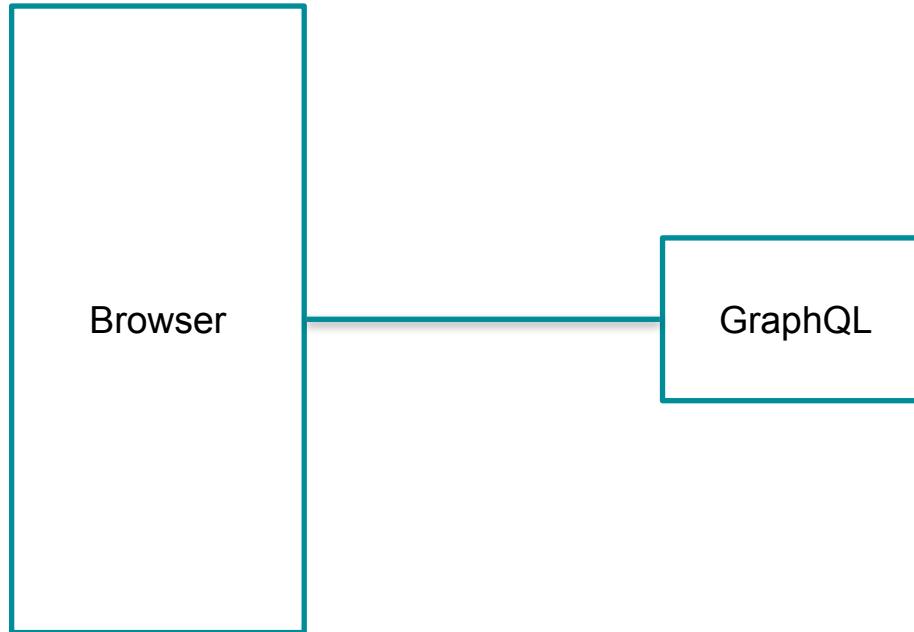


jasonrclark

# REST



# GraphQL



# <https://spec.graphql.org/October2021/>

A screenshot of a web browser window displaying the GraphQL specification. The title bar says "GraphQL". The address bar shows the URL "spec.graphql.org/October2021/". The main content area has a large "GraphQL" logo at the top, followed by the text "October 2021 Edition". Below that is a section titled "Introduction" with a detailed paragraph about the specification's history and licensing.

**GraphQL**

*October 2021 Edition*

**Introduction**

This is the specification for GraphQL, a query language and execution engine originally created at Facebook in 2012 for describing the capabilities and requirements of data models for client-server applications. The development of this open standard started in 2015. This specification was licensed under OWFa 1.0 in 2017. The [GraphQL Foundation](#) was formed in 2019 as a neutral focal point for organizations who support the GraphQL ecosystem, and the [GraphQL Specification Project](#) was established also in 2019 as the Joint Development Foundation Projects, LLC, GraphQL Series.

# Type Definitions



@jasonrclark@ruby.social



jasonrclark

# Scalars

Int

Float

String

Boolean

ID



@jasonrclark@ruby.social



jasonrclark

# Objects

```
type Movie {  
  id: ID  
  title: String  
  budget: Int  
}
```



@jasonrclark@ruby.social



jasonrclark

# Custom scalars

**scalar DateTime**

```
type Movie {  
    id: ID  
    title: String  
    budget: Int  
    premier: DateTime  
}
```



@jasonrclark@ruby.social



jasonrclark

# Lists

```
type Movie {  
    id: ID  
    title: String  
    budget: Int  
    premier: DateTime  
    reviews: [Review]  
}
```

```
type Review {  
    score: Int  
    opinion: String  
}
```



@jasonrclark@ruby.social



jasonrclark

# Arguments

```
type Movie {  
    ...  
    reviews(score: Int): [Review]  
}  
  
type Review {  
    score: Int  
    opinion: String  
}
```



@jasonrclark@ruby.social



jasonrclark

# Arguments

```
type Movie {  
    ...  
    reviews(score: Int!): [Review]  
}
```

Required  
↓

```
type Review {  
    score: Int  
    opinion: String  
}
```



@jasonrclark@ruby.social



jasonrclark

# Enums

```
type Review {  
  ...  
  sentiment: Sentiment  
}  
  
enum Sentiment {  
  POSITIVE,  
  NEUTRAL,  
  NEGATIVE  
}
```

# Query and Mutation



@jasonrclark@ruby.social



jasonrclark

# Querying over HTTP

```
curl http://localhost:8080/graphql \
-H 'content-type: application/graphql' \
--data 'query { ... }'
```

```
{"data": { ... }}
```



@jasonrclark@ruby.social



jasonrclark

# Data fetching

```
type Query {  
  movie(id: ID!): Movie  
}
```

```
type Movie {  
  id: ID  
  title: String  
  budget: Int  
  release: DateTime  
  reviews: [Reviews]  
}
```



@jasonrclark@ruby.social



jasonrclark

# JSON Results

```
query{  
  movie(id:"1") {  
    id  
    title  
    budget  
    release  
  }  
}
```

```
{  
  "data": {  
    "movie": {  
      "id": "1",  
      "title": "her",  
      "budget": 23000,  
      "release": "2014"  
    }  
  }  
}
```

# JSON Results

```
query{  
  movie(id:"1") {  
    id  
    title  
  }  
}
```

```
{  
  "data": {  
    "movie": {  
      "id": "1",  
      "title": "her"  
    }  
  }  
}
```



# Data writing

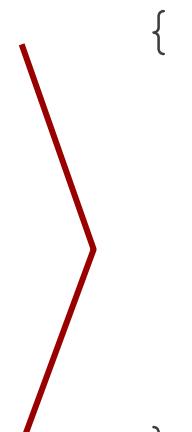
```
type Mutation {  
  reviewMovie(review: ReviewInput!): Movie  
}  
  
input ReviewInput {  
  movieId: ID!  
  score: Int!  
  opinion: String  
}
```

# Writing data

```
type Mutation {  
  reviewMovie(review: ReviewInput!): Movie  
}  
  
input ReviewInput {  
  movieId: ID!  
  score: Int!  
  opinion: String  
}
```

# Mutating

```
mutation {  
  reviewMovie (title: "her") {  
    review: {  
      movieId: "1",  
      score: 4  
    }  
  }  
}
```



```
{  
  "data": {  
    "reviewMovie": {  
      "title": "her"  
    }  
  }  
}
```

# First-class errors



@jasonrclark@ruby.social



jasonrclark

# Errors

```
{  
  "errors": [  
    {  
      "message": "Movie not found",  
      "locations": [ { "line": 6, ... } ],  
      "path": [ "movie" ]  
    }  
  ],  
  "data": null  
}
```



@jasonrclark@ruby.social



jasonrclark

# Extending errors

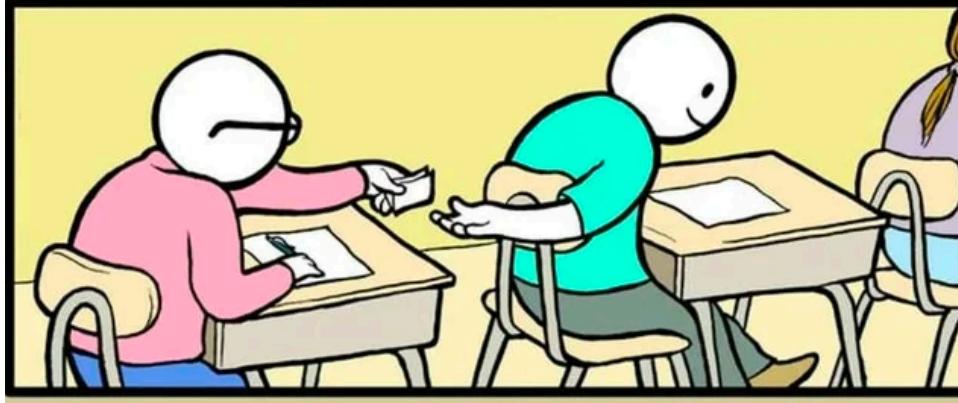
```
{  
  "errors": [  
    {  
      "message": "Movie not found",  
      "path": ["movie"],  
      "extensions": {  
        "code": "MOVIE_NOT_FOUND"  
      }  
    }  
  ],  
  "data": null  
}
```



@jasonrclark@ruby.social



jasonrclark



@jasonrclark@ruby.social



jasonrclark

# Errors and data

```
{  
  "errors": [  
    {  
      "message": "Movie budget not found",  
      "path": ["movie", "budget"]  
    } ],  
  "data": {  
    "movie": {  
      "title": "her"  
      "budget": null  
    }  
  }  
}
```

And a lot more....

A circular word cloud centered around the word "Fragments". Other prominent words include "Subscriptions", "Arguments", "Interfaces", and "Types". Smaller words surrounding the center include "Fields", "Execution", "Introspection", "Operation", "Graphs", "Schemas", "Mutations", "Variables", "Inline", "Caching", "Aliases", "Union", "Non-Null", "Lists", "Input", "Object", "Validation", "Authorization", "Directives", "Batching", "Scalar", "Queries", "Pagination", and "Enumeration".



@jasonrclark@ruby.social



jasonrclark

# Why choose GraphQL?



@jasonrclark@ruby.social



jasonrclark

# Reduces the number of requests

Smaller response payloads

Promotes continuous growth

Avoids the explosion of endpoints

Self-documented

Reduces the number of requests

## **Smaller response payloads**

Promotes continuous growth

Avoids the explosion of endpoints

Self-documented

Reduces the number of requests

Smaller response payloads

## Promotes continuous growth

Avoids the explosion of endpoints

Self-documented



@jasonrclark@ruby.social



jasonrclark

Reduces the number of requests

Smaller response payloads

Promotes continuous growth

## Avoids the explosion of endpoints

Self-documented

Reduces the number of requests

Smaller response payloads

Promotes continuous growth

Avoids the explosion of endpoints

**Self-documented**

# How to build it?



@jasonrclark@ruby.social



jasonrclark

# graphql-java



@jasonrclark@ruby.social



jasonrclark

# <https://www.graphql-java.com/documentation/getting-started>

The screenshot shows a web browser displaying the [GraphQL Java Documentation](https://www.graphql-java.com/documentation/getting-started). The page is titled "Getting started" and is version v20. The left sidebar lists various documentation topics, and the main content area provides instructions for using the latest release with Gradle and Maven.

**Getting started**

Version: v20

# Getting started

graphql-java requires at least Java 8.

## How to use the latest release with Gradle

Make sure `mavenCentral` is among your repos:

[Gradle](#) [Gradle \(Kotlin\)](#)

```
repositories {  
    mavenCentral()  
}  
  
dependencies {  
    implementation 'com.graphql-java:graphql-java:20.2'  
}
```

## How to use the latest release with Maven

Dependency:

```
<dependency>  
    <groupId>com.graphql-java</groupId>  
    <artifactId>graphql-java</artifactId>  
    <version>20.2</version>  
</dependency>
```

# Serving over HTTP

Wiring up the schema

Fetching data

Don't Repeat Yourself



@jasonrclark@ruby.social

jasonrclark

<https://graphql.github.io/graphql-over-http/draft/>

A screenshot of a web browser window displaying the "GraphQL Over HTTP" draft specification. The title bar shows the URL: graphql.github.io/graphql-over-http/draft/. The main content area features a large heading "GraphQL Over HTTP". Below it is a note card with the word "Note" and a small icon. The note text reads: "Stage 1: Proposal This spec is under active development, and not ready for implementations yet. For more information, please see the [Roadmap](#) or [how to get involved](#). You can find our community in the #graphql-over-http channel on the [GraphQL Foundation Discord](#)." At the bottom of the note card is a separator line with three dashes. The footer contains a "Contents" section with links to various parts of the specification.

**Contents**

- [1 GraphQL over HTTP](#)
- [2 Overview](#)
- [3 URL](#)
- [4 Serialization Format](#)
  - [4.1 Media Types](#)
- [5 Request](#)

# <https://github.com/spring-projects/spring-graphql>

The screenshot shows the GitHub repository page for `spring-projects/spring-graphql`. The repository is public, has 43 watchers, 234 forks, and 1.3k stars. It contains 3 branches and 19 tags. The main branch is selected. The repository's purpose is described as "Spring Integration for GraphQL". It includes links to the project's website (`spring.io/projects/spring-graphql`), GitHub pages (`graphql` and `spring-graphql`), and various documentation and policy links. A list of recent commits is displayed, showing contributions from `rstoyanchev` and others, including updates to QueryDSL, QBE, and various build and dependency upgrades.

Code

Issues 36

Pull requests 5

Actions

Wiki

Security

Insights

main

3 branches

19 tags

Go to file

Add file

Code

**About**

Spring Integration for GraphQL

[spring.io/projects/spring-graphql](https://spring.io/projects/spring-graphql)

graphql

spring-graphql

Readme

Apache-2.0 license

Code of conduct

Security policy

1.3k stars

43 watching

234 forks

Releases 15

# Dropwizard?



@jasonrclark@ruby.social



jasonrclark

```
// Dropwizard

@Provides
@IntoSet
@Singleton
@Named("jerseyComponents")
static Object resource(GraphQL graphql) {
    return new GraphQLResource(graphql);
}
```



@jasonrclark@ruby.social



jasonrclark

# GraphQLResource

```
@Singleton  
@Path("/")  
@Produces(MediaType.APPLICATION_JSON)  
public class GraphQLResource {  
  
    private final GraphQL graphql;  
  
    @Inject  
    public GraphQLResource(GraphQL graphql) {  
        this.graphql = graphql;  
    }  
}
```



# GraphQLResource

```
@GET  
@Path("graphql")  
public Response getGraphql(  
    @Context HttpHeaders headers,  
    @QueryParam("query") String query,  
    @QueryParam("operationName") String op,  
    @QueryParam("variables") String vars) {  
    var request = new GraphQLRequest();  
    var context = new RequestContext(...);  
    return execute(...);  
}
```



# Serving over HTTP

## Wiring up the schema

Fetching data

Don't Repeat Yourself



@jasonrclark@ruby.social



jasonrclark

# newrelic-graphql-java-core



@jasonrclark@ruby.social



jasonrclark

# <https://github.com/newrelic/newrelic-graphql-java-core>

The screenshot shows a GitHub repository page for the project "newrelic-graphql-java-core". The page includes the README.md file, which features an "Open Source" badge with the New Relic logo, a "Community project" badge, and a note that it is an official New Relic open source project. The "Languages" section indicates Java is the primary language at 94.5%, followed by Kotlin at 5.2% and HTML at 0.3%. The "Getting Started" section provides Maven and Gradle configuration snippets.

newrelic-graphql-java-core

This library provides helpers for integrating [graphql-java](#) with your JVM application. These simplify schema configuration and input mapping, and are in active use for GraphQL services at New Relic.

## Getting Started

### Maven

```
<dependency>
<groupId>com.newrelic.graphql</groupId>
<artifactId>core</artifactId>
<version>0.1.1</version>
</dependency>
```

### Gradle

```
compile("com.newrelic.graphql:core:0.1.1")
```

```
// newrelic-graphql-java-core

new SimpleGraphQLBuilder(schemaReader)
    .fetcher("Query", "movie",
              new QueryMovie())
    .fetcher("Movie", "reviews",
              new QueryReviews())
    .fetcher("Mutation", "reviewMovie",
              new MutationReviewMovie())
    .build();
```



@jasonrclark@ruby.social



jasonrclark

## // newrelic-graphql-java

```
new SimpleGraphQLBuilder(schemaReader)
    .fetcher("Query", "movie",
              new QueryMovie())
    .fetcher("Movie", "reviews",
              new QueryReviews())
    .fetcher("Mutation", "reviewMovie",
              new MutationReviewMovie())
    .build();
```



@jasonrclark@ruby.social



jasonrclark

```
// newrelic-graphql-java

new SimpleGraphQLBuilder(schemaReader)
    .fetcher("Query", "movie",
              new QueryMovie())
    .fetcher("Movie", "reviews",
              new QueryReviews())
    .fetcher("Mutation", "reviewMovie",
              new MutationReviewMovie())
    .build();
```



@jasonrclark@ruby.social



jasonrclark

```
// newrelic-graphql-java

new SimpleGraphQLBuilder(schemaReader)
    .fetcher("Query", "movie",
              new QueryMovie())
    .fetcher("Movie", "reviews",
              new QueryReviews())
    .fetcher("Mutation", "reviewMovie",
              new MutationReviewMovie())
    .build();
```



@jasonrclark@ruby.social



jasonrclark

# Serving over HTTP

Wiring up the schema

## Fetching data

| Don't Repeat Yourself



@jasonrclark@ruby.social



jasonrclark

## // Our data objects

```
public class Movie {  
  
    public Movie(int id, String title,  
                int budget, String premier) {  
        this.id = id;  
        this.title = title;  
        this.budget = budget;  
        this.premier = premier;  
    }  
  
    public int getId() {  
        return id;  
    }  
  
    ...
```

// Our fetcher

```
public class QueryMovie
  implements DataFetcher<Movie> {

  @Override
  public Movie get(DataFetchingEnvironment environment) {
    Integer id = Integer.parseInt(
      environment.getArgument("id"));
    return MovieService.find(id);
  }
}
```



@jasonrclark@ruby.social



jasonrclark

// Our fetcher

```
public class QueryMovie
  implements DataFetcher<Movie> {

  @Override
  public Movie get(DataFetchingEnvironment environment) {
    Integer id = Integer.parseInt(
      environment.getArgument("id"));
    return MovieService.find(id);
  }
}
```



@jasonrclark@ruby.social



jasonrclark

```
// Our fetcher

public class QueryMovie
    implements DataFetcher<Movie> {

    @Override
    public Movie get(DataFetchingEnvironment environment) {
        Integer id = Integer.parseInt(
            environment.getArgument("id"));
        return MovieService.find(id);
    }
}
```



@jasonrclark@ruby.social



jasonrclark

```
// Our fetcher
```

```
public class QueryMovie
  implements DataFetcher<Movie> {

  @Override
  public Movie get(DataFetchingEnvironment environment) {
    Integer id = Integer.parseInt(
      environment.getArgument("id"));
    return MovieService.find(id);
  }
}
```



@jasonrclark@ruby.social



jasonrclark

# Serving over HTTP

## Wiring up the schema

## Fetching data



## Don't Repeat Yourself



@jasonrclark@ruby.social



jasonrclark

# Said our types already

```
type Movie {  
    id: ID  
    title: String  
    budget: Int  
    premier: DateTime  
    reviews: [Review]  
}
```

```
type Review {  
    score: Int  
    opinion: String  
}
```

# `newrelic-graphql-api-gen-plugin`



@jasonrclark@ruby.social



jasonrclark

# Generated Code

```
▼ graphql-from-the-ground-up ~/source/newrelic/graphql-from-the-ground-up
  ► .gradle
  ► .idea
  ► build
  ► gradle
  ► out
  ▼ src
    ► dist [main]
    ▼ main
      ▼ java
        ▼ com.newrelic
          graphqlservice
          ▼ graphqlservice
            ► fetchers
            ▼ model
              c DateTime
              c Movie
              c Mutation
              c Query
              c Review
              c ReviewInput
              E Sentiment
```



@jasonrclark@ruby.social



jasonrclark

# Schema knows the arguments

```
type Query {  
    movie(id: ID!): Movie  
}
```



@jasonrclark@ruby.social



jasonrclark

```
// Our fetcher

public class QueryMovieFetcher
  extends Query.QueryMovieBase {
    @Override
    public Movie get(
      DataFetchingEnvironment environment,
      String id) {
      return MovieService.find(Integer.parseInt(id));
    }
}
```



@jasonrclark@ruby.social



jasonrclark

```
// Our mutation
```

```
public class MutationReviewMovieFetcher  
    implements DataFetcher<Movie> {  
  
    @Override  
    public Movie get(DataFetchingEnvironment environment) {  
        var rawInput = environment.getArgument("review");  
  
        ReviewInput input = new ReviewInput(  
            Integer.parseInt(rawInput.get("movieId"))...,  
            Integer.parseInt(rawInput.get("score"))...,  
            rawInput.get("opinion").toString()  
        );  
        ReviewService.saveReview(input);  
  
        return MovieService.find(input.getMovieId());  
    }  
}
```



@jasonrclark@ruby.social



jasonrclark

```
// Our mutation
```

```
public class MutationReviewMovieFetcher
    implements DataFetcher<Movie> {

    @Override
    public Movie get(DataFetchingEnvironment environment) {
        var rawInput = environment.getArgument("review");

        ReviewInput input = new ReviewInput(
            Integer.parseInt(rawInput.get("movieId")),
            Integer.parseInt(rawInput.get("score")),
            rawInput.get("opinion").toString()
        );
        ReviewService.saveReview(input);

        return MovieService.find(input.getMovieId());
    }
}
```



@jasonrclark@ruby.social



jasonrclark

// Our shiny new code

```
public class MutationReviewMovieFetcher
  extends Mutation.MutationReviewMovieBase {
    @Override
    public Movie get(DataFetchingEnvironment environment,
                     ReviewInput input) {
      ReviewService.saveReview(input);
      return MovieService.find(
        Integer.parseInt(input.getMovieId())));
    }
}
```

# How did it go?

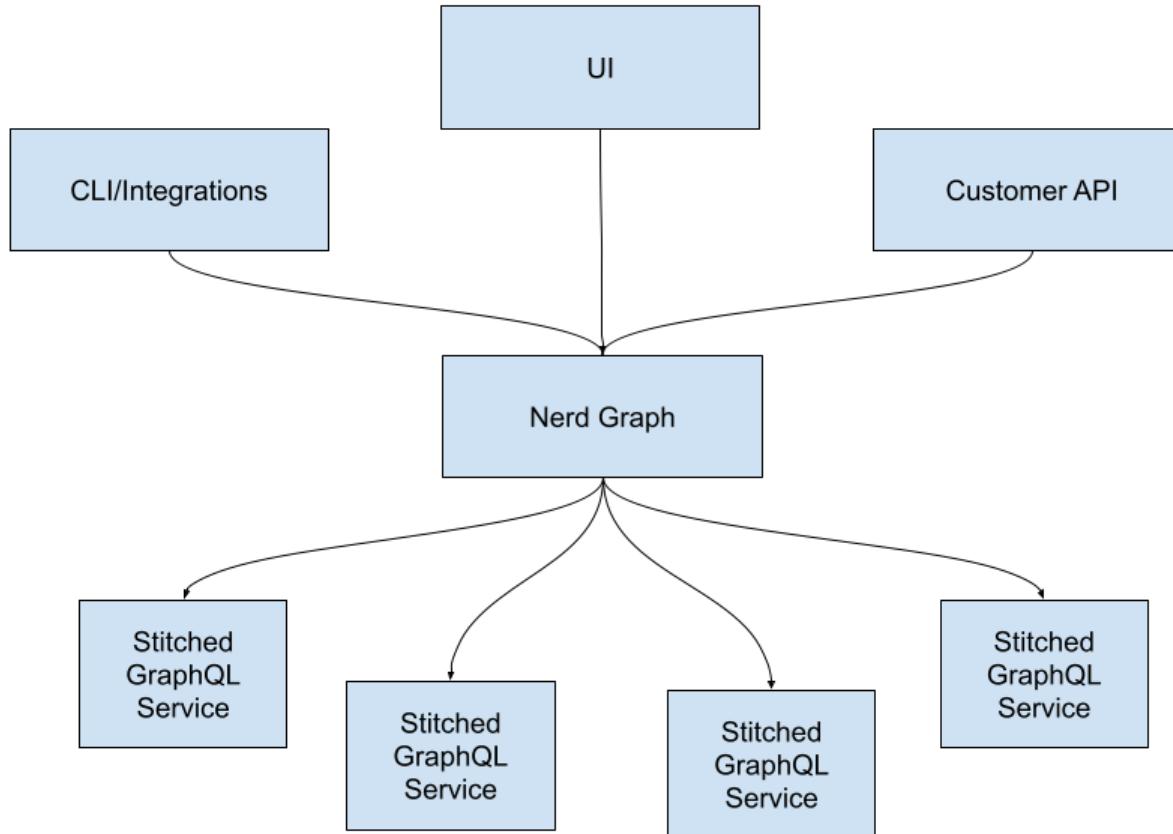


@jasonrclark@ruby.social



jasonrclark

# Federated GraphQL



@jasonrclark@ruby.social



jasonrclark

# Embracing the constraints

Evolving schemas

Finding common patterns



@jasonrclark@ruby.social



jasonrclark

# Errors like HTTP

```
{  
  "data": {  
    "movie": {  
      "status": {  
        "code": 403  
      }  
    }  
  }  
}
```

# Use what GraphQL Provides instead

```
{  
  "errors": [  
    {  
      "message": "Not authorized to see movie",  
      "path": ["movie"],  
      "extensions": {  
        "code": "NOT_AUTHORIZED"  
      }  
    },  
    "data": null  
  }  
}
```



@jasonrclark@ruby.social



jasonrclark

# Keeping the Types

## **scalar JSON**

```
type Query {  
    attributes: JSON  
}
```



@jasonrclark@ruby.social



jasonrclark

# Keeping the Types

**scalar CustomAttributes**

```
type Query {  
    attributes: CustomAttributes  
}
```



@jasonrclark@ruby.social



jasonrclark

Embracing the constraints

## Evolving schemas

Dealing with fetcher interactions



@jasonrclark@ruby.social



jasonrclark

# Evolving schema

- Let go of strict versioning
  - Design for addition
  - Required vs optional
  - One vs many
- Automated schema linting
- Instrumentation to guide deprecation



@jasonrclark@ruby.social



jasonrclark

Embracing error handling

Evolving our schemas

## Finding common patterns



@jasonrclark@ruby.social



jasonrclark

# Finding common patterns

- With company-wide, where do we let teams extend?
- Pagination of results
- Safe error handling (internal vs external details)



@jasonrclark@ruby.social



jasonrclark

# Where We've Been Today

- What's GraphQL?
- Why choose GraphQL?
- How to build it?
- How did it go?



@jasonrclark@ruby.social



jasonrclark

# Questions?

