GraphQL from the Ground Up

Jason Clark
New Relic, Principal Engineer and Architect

With special thanks to Irene Lopez

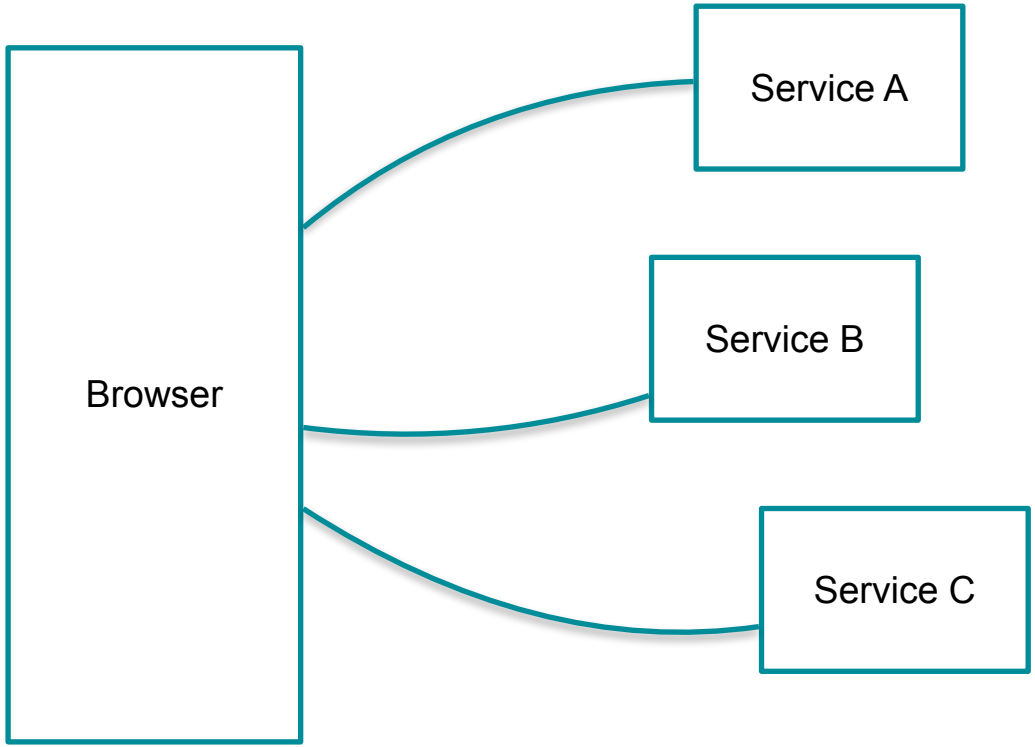
@jasonrclark@ruby.social jasonrclark

Where We're Going Today

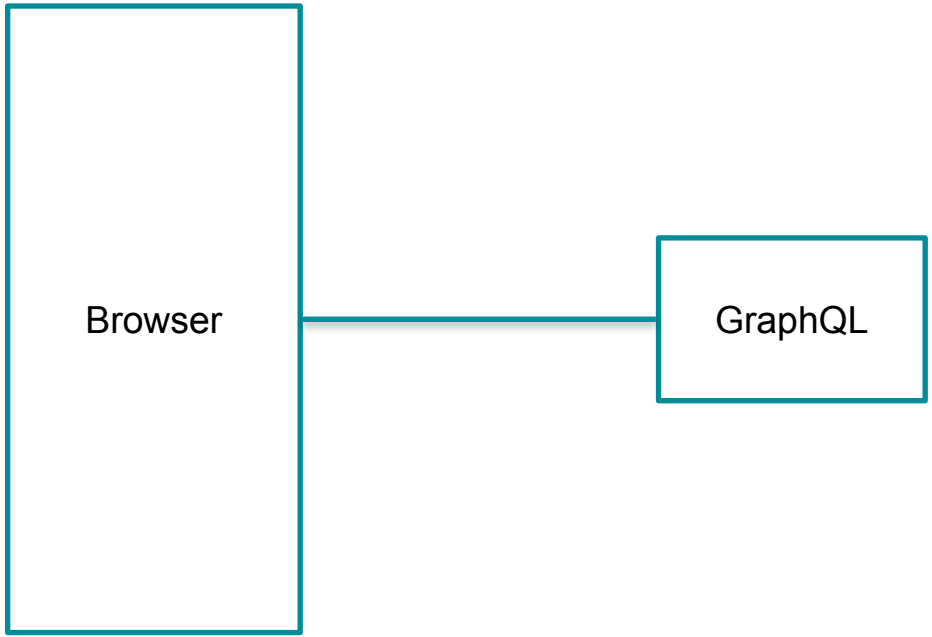
- What's GraphQL?
- Why choose GraphQL?
- How to build it?
- How did it go?

What’s GraphQL?

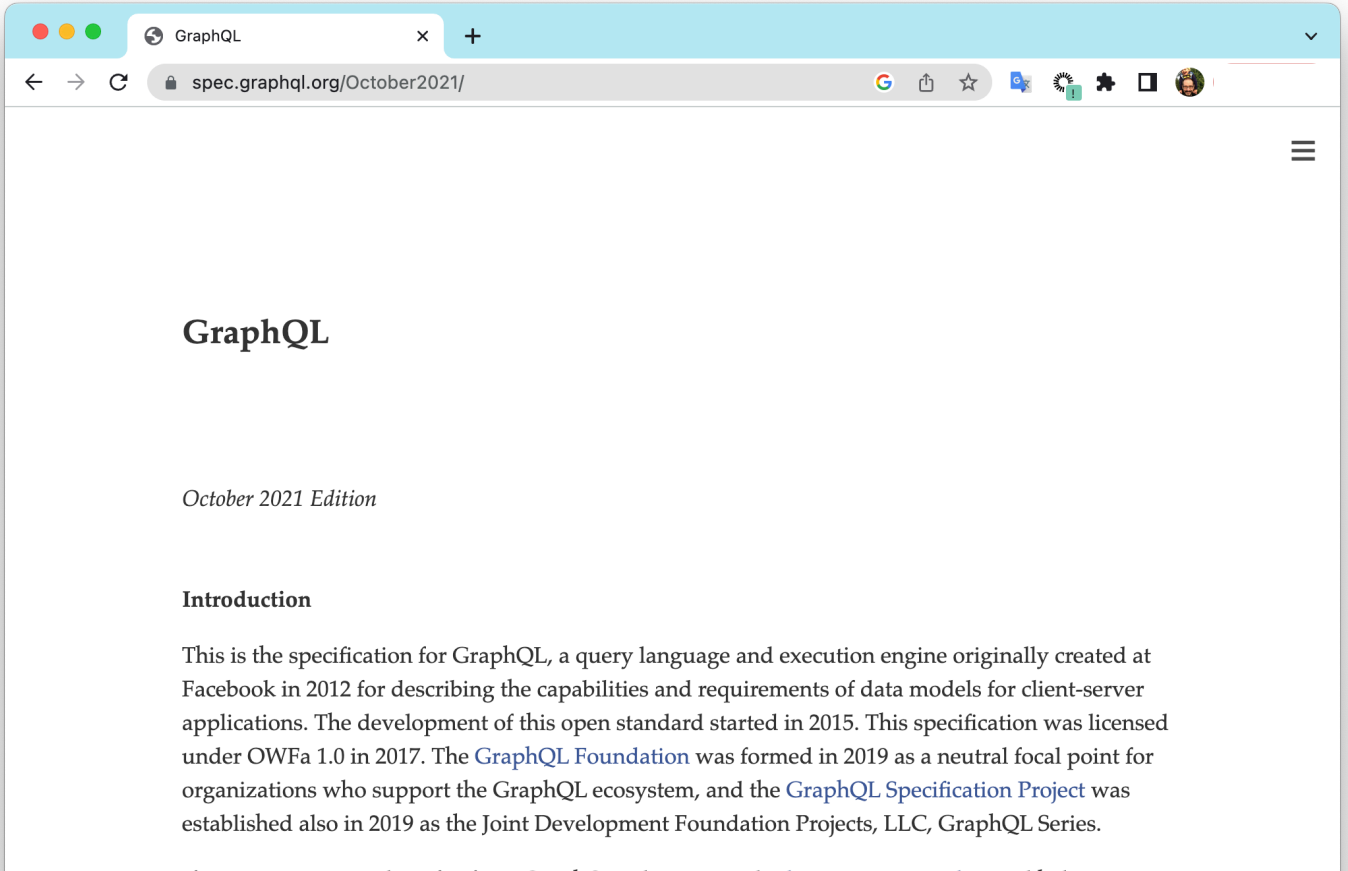
REST



GraphQL



https://spec.graphql.org/October2021/



Type Definitions

Scalars

```
Int
Float
String
Boolean
ID
```

Objects

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

Custom scalars

```
scalar DateTime

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

Lists

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

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

Arguments

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

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

Arguments

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

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

Enums

```
type Review {
  ...
  sentiment: Sentiment
}

enum Sentiment {
  POSITIVE,
  NEUTRAL,
  NEGATIVE
}
```

Query and Mutation

Querying over HTTP

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

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

Data fetching

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

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

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

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

First-class errors

Errors

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

Extending errors

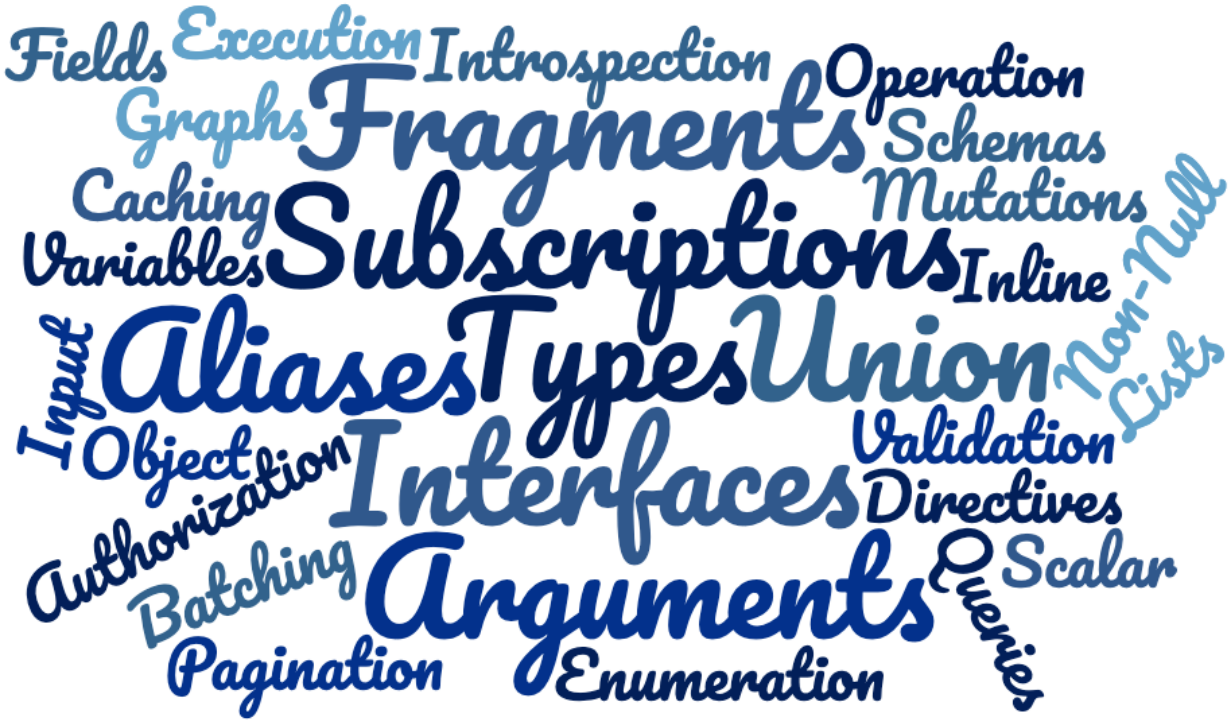
```
{
  "errors": [
    {
      "message": "Movie not found",
      "path": ["movie"],
      "extensions": {
        "code": "MOVIE_NOT_FOUND"
      }
    }
  ],
  "data": null
}
```


HTTP 200
{
"status": 404
"msg": "Not Found"
}

Errors and data

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

And a lot more....



Why choose GraphQL?

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

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

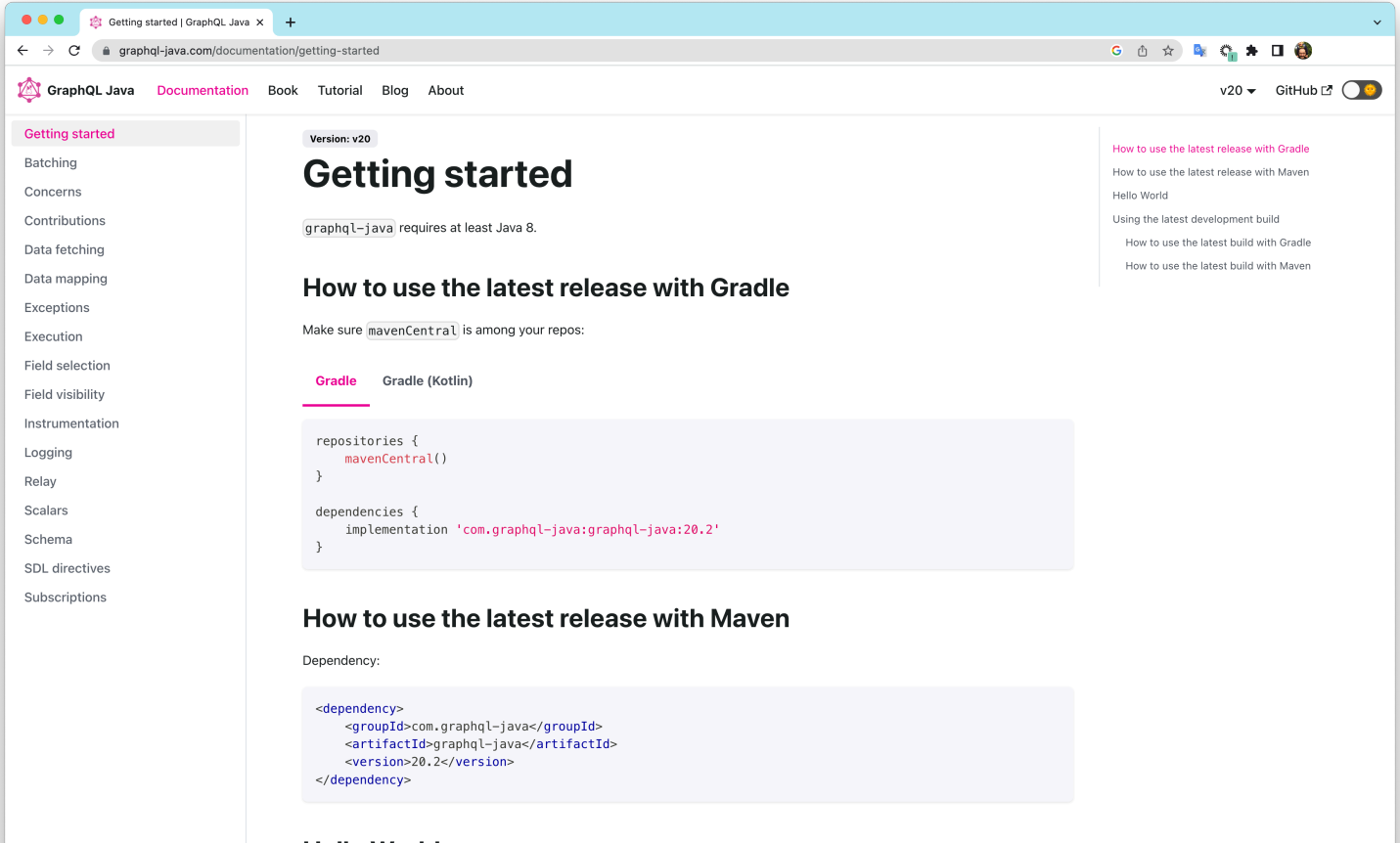
Self-documented

How to build it?

graphql-java

@jasonrclark@ruby.social jasonrclark

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



jasonrclark

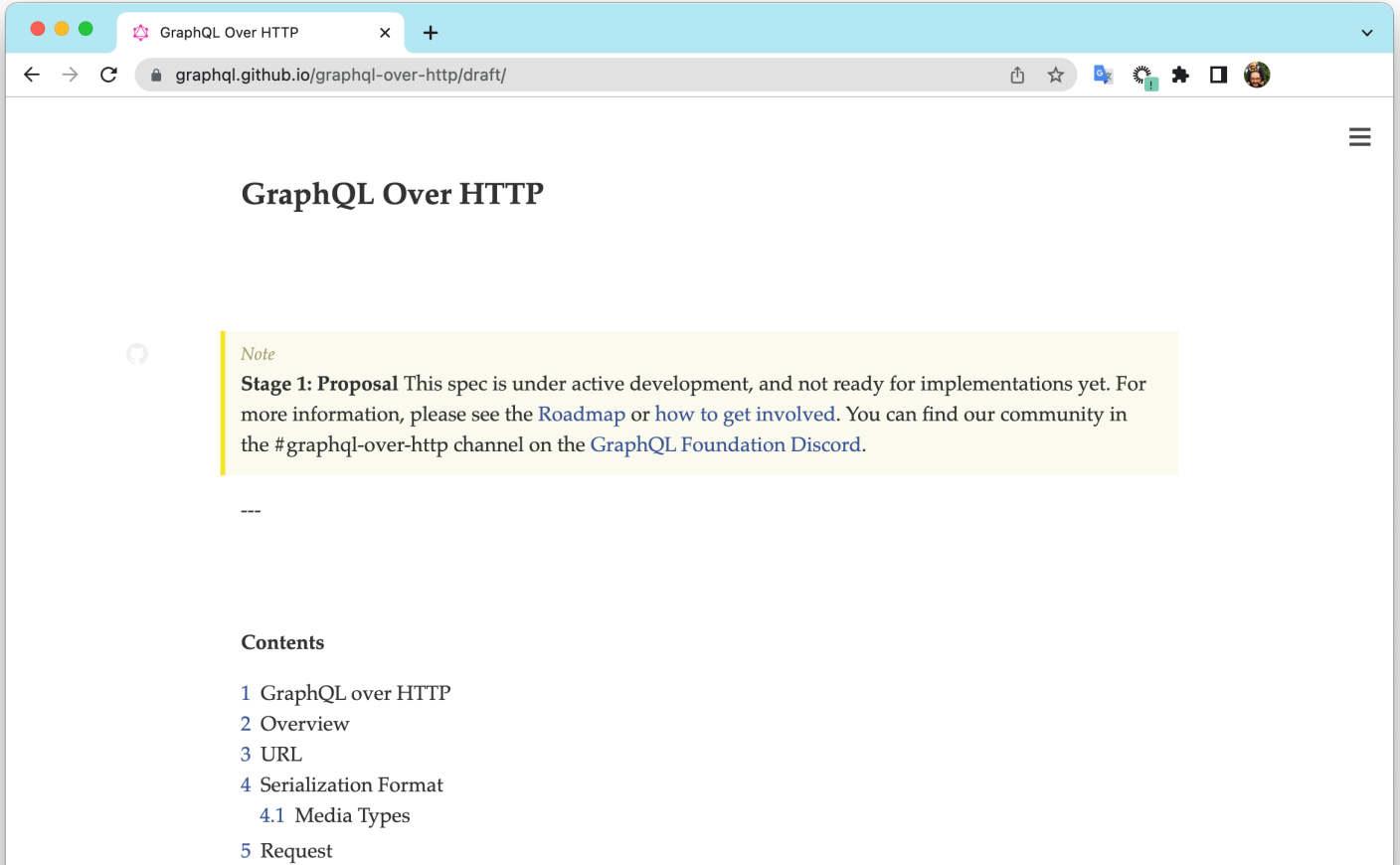
Serving over HTTP

Wiring up the schema

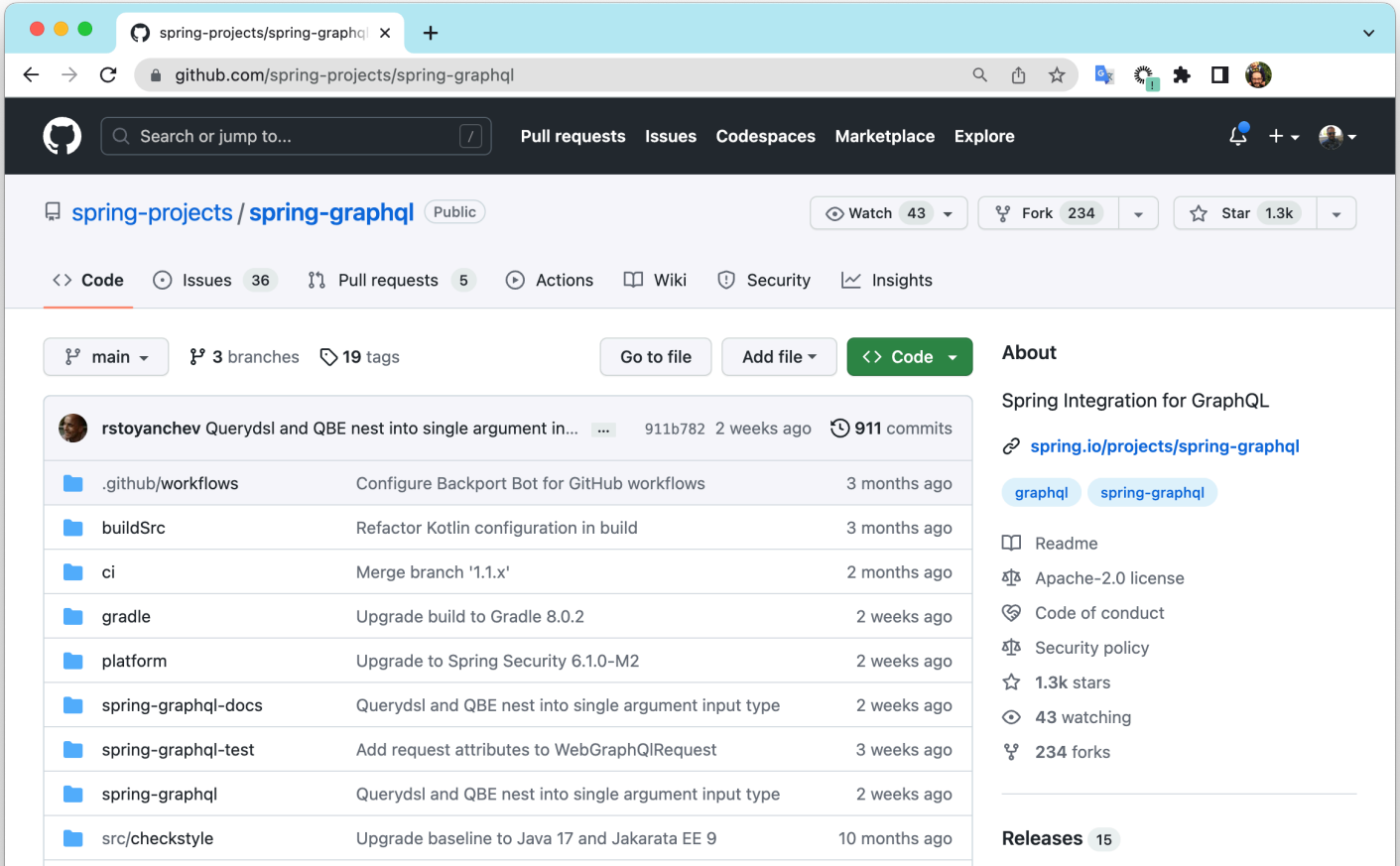
Fetching data

Don't Repeat Yourself

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



https://github.com/spring-projects/spring-graphql



Dropwizard?

```
// Dropwizard

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

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

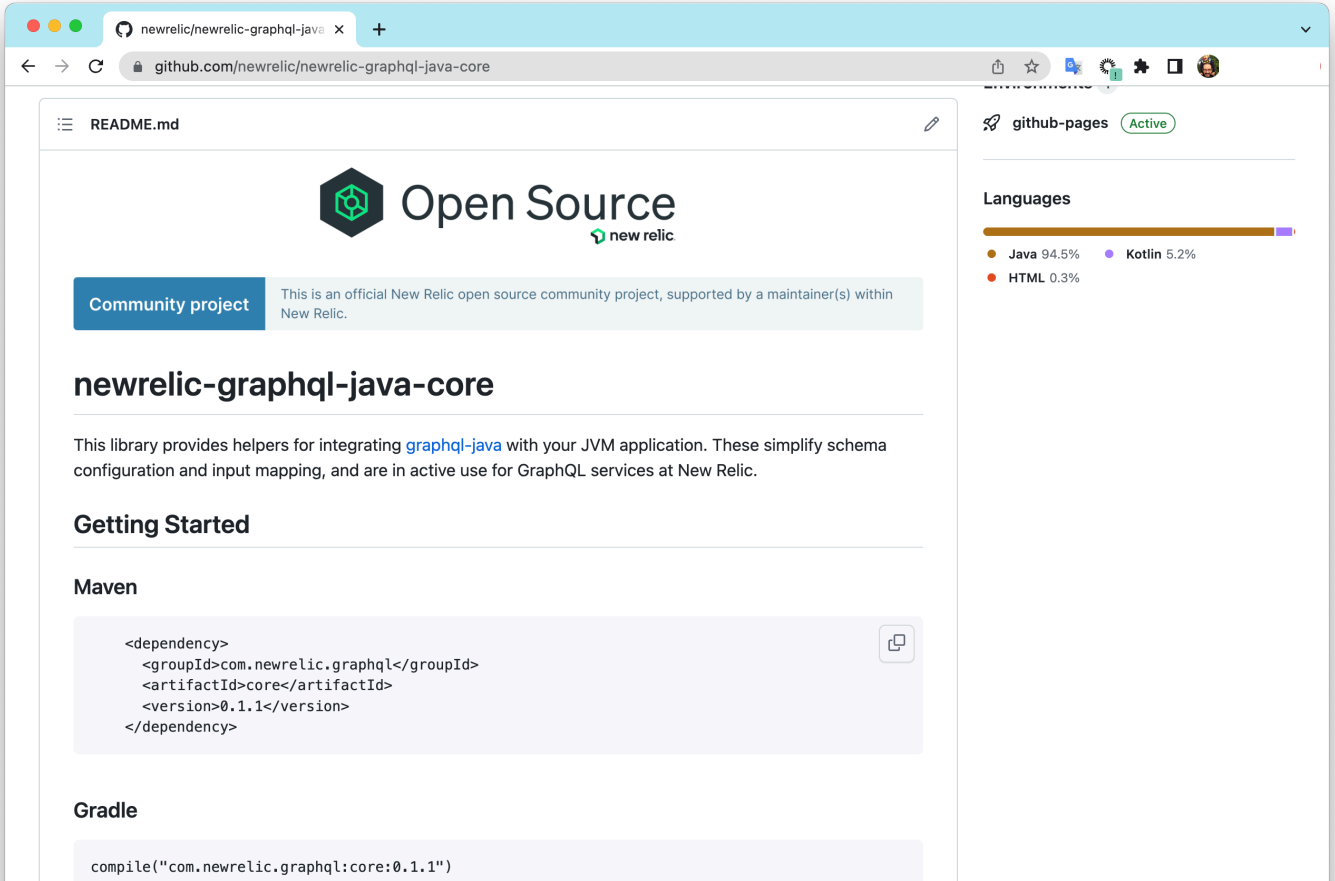
Wiring up the schema

Fetching data

Don't Repeat Yourself

newrelic-graphql-java-core

https://github.com/newrelic/newrelic-graphql-java-core



jasonrclark

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

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

```
// newrelic-graphql-java

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

```
// newrelic-graphql-java

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

```
// newrelic-graphql-java

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

Serving over HTTP

Wiring up the schema

Fetching data

Don't Repeat Yourself

```
// 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);
  }
}
```

```
// 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);
  }
}
```

```
// 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);
  }
}
```

```
// 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);
  }
}
```

Serving over HTTP

Wiring up the schema

Fetching data

Don't Repeat Yourself

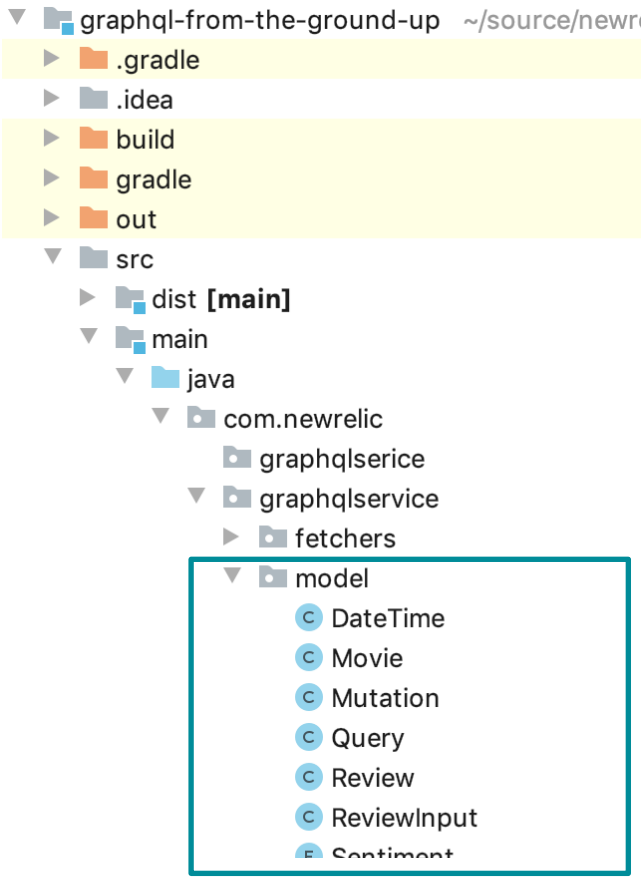
Said our types already

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

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

`newrelic-graphql-api-gen-plugin`

Generated Code



Schema knows the arguments

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

```
// Our fetcher

public class QueryMovieFetcher
    extends Query.QueryMovieBase {

    @Override
    public Movie get(
        DataFetchingEnvironment environment,
        String id) {

        return MovieService.find(Integer.parseInt(id));
    }
}
```

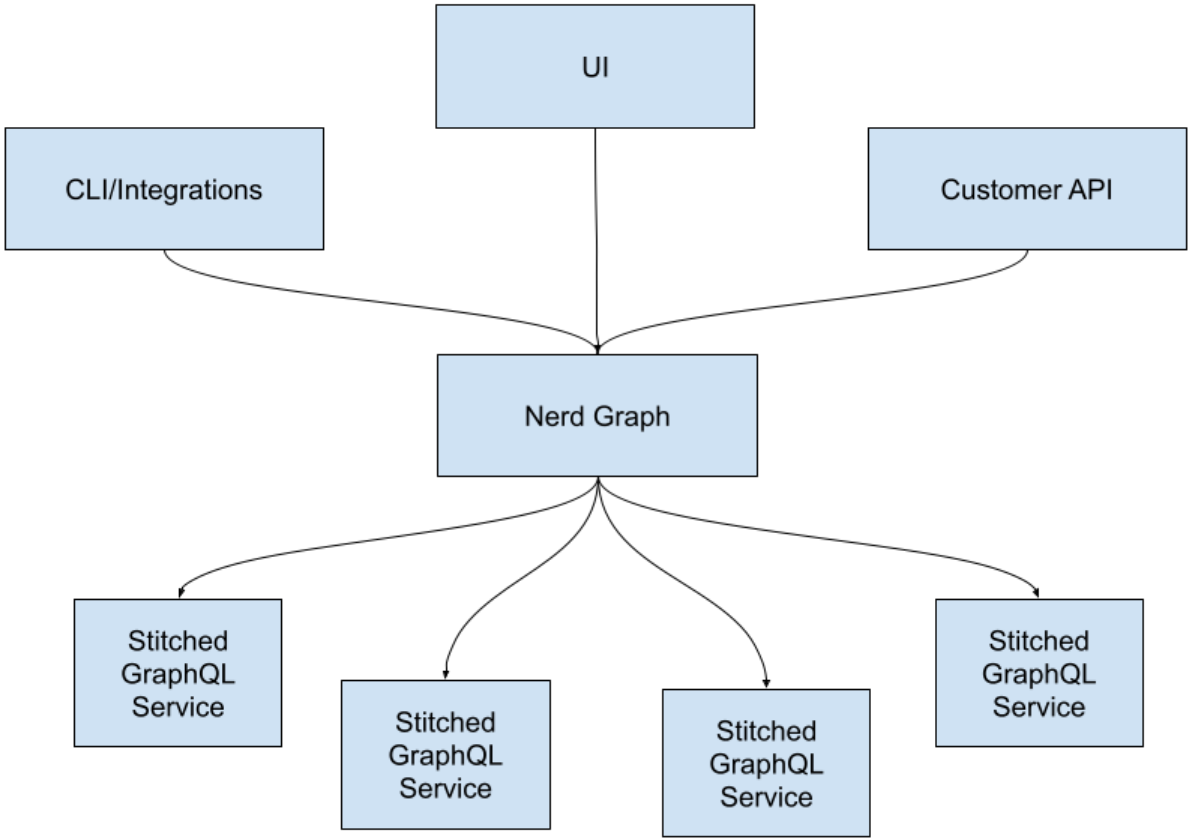
```
// Our mutation

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());
    }
}
```

```
// 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());
    }
}
```

```
// 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?

Federated GraphQL



Embracing the constraints

Evolving schemas

Finding common patterns

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
}
```

Keeping the Types

```
scalar JSON

type Query {
   attributes: JSON
}
```

Keeping the Types

```
scalar CustomAttributes

type Query {
   attributes: CustomAttributes
}
```

Embracing the constraints

Evolving schemas

Dealing with fetcher interactions

Evolving schema

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

Embracing error handling

Evolving our schemas

Finding common patterns

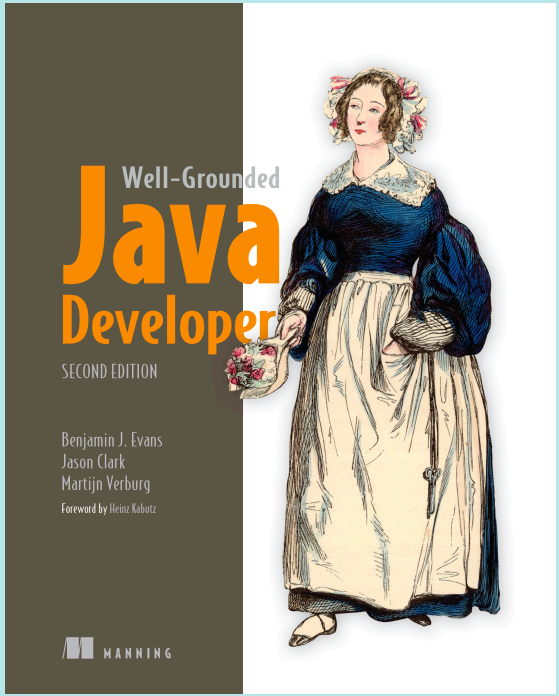
Finding common patterns

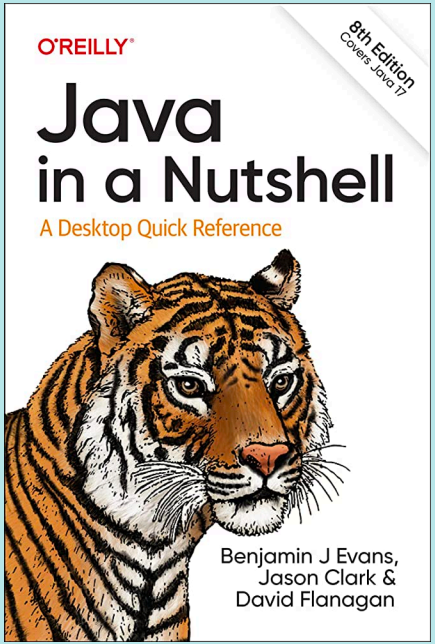
- With company-wide, where do we let teams extend?
- Pagination of results
- Safe error handling (internal vs external details)

Where We've Been Today

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

Questions?





@jasonrclark@ruby.social jasonrclark