

Full Stack Development With

GraphQL + Neo4j



William Lyon
@lyonwj
lyonwj.com



GraphConnect
Powered by Neo4j



William Lyon

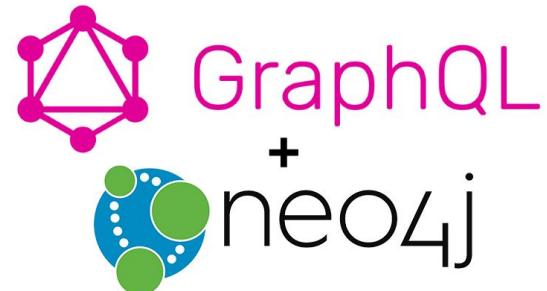
DevRel Engineering @neo4j

will@neo4j.com
[@lyonwj](https://twitter.com/lyonwj)
lyonwj.com



GraphQL + Neo4j

- An overview of GraphQL
- Building a GraphQL service
- Neo4j-GraphQL integration(s)



An Overview Of GraphQL

- A new paradigm for building APIs
- Schema definition
 - Types
 - GraphQL entry points (Query & Mutation types)
- Query language for APIs
 - Limited support for “queries” (aggregations, filtering, ...)
- Community of tools
 - GraphiQL
 - Mocking
 - Performance monitoring



GraphQL

GraphiQL



Prettify

History

Query

Movie



```
1 query MovieSearch($title: String!, $limit: Int!) {  
2   movies(subString:$title, limit:$limit) {  
3     movieId  
4     title  
5     year  
6     genres  
7     poster  
8     plot  
9     imdbRating  
10    similar {  
11      title  
12      poster  
13      year  
14    }  
15  }  
16}  
17}
```

```
{  
  "data": {  
    "movies": [  
      {  
        "movieId": "3100",  
        "title": "River Runs Through It, A",  
        "year": 1992,  
        "genres": [  
          "Drama"  
        ],  
        "poster": "http://ia.media-imdb.com/images/M/MV5BMTM2Nzc5MjI4NF5BMl5BanBnXkFtZTYwNzgwMjc5._V1_SX300.jpg",  
        "plot": "The story about two sons of a stern minister -- one reserved, one rebellious -- growing up in rural Montana while devoted to fly fishing.",  
        "imdbRating": 7.3,  
        "similar": [  
          {  
            "title": "Forrest Gump",  
            "poster": "http://ia.media-imdb.com/images/M/MV5BMTI1Nzk1MzQwMV5BMl5BanBnXkFtZTYwODkxOTA5._V1_SX300.jpg",  
            "year": 1994  
          },  
          {  
            "title": "Titanic",  
            "poster": "http://ia.media-imdb.com/images/M/MV5BMjExNzM0NDM0N15BMl5BanBnXkFtZTcwMzkxOTUwNw@@._V1_SX300.jpg",  
            "year": 1997  
          },  
          {  
            "title": "Shawshank Redemption, The",  
            "poster": "http://ia.media-imdb.com/images/M/MV5BODU4MjU4NjIwNl5BMl5BanBnXkFtZTgwMDU2MjEyMDE@._V1_SX300.jpg",  
            "year": 1994  
          }  
        ]  
      }  
    ]  
  }  
}
```

QUERY VARIABLES

```
1 {  
2   "title": "River Runs Through It",  
3   "limit": 3  
4 }
```

Search Movie...

No Description

FIELDS

movieId: String!

title: String

year: Int

plot: String

poster: String

imdbRating: Float

genres: [String]

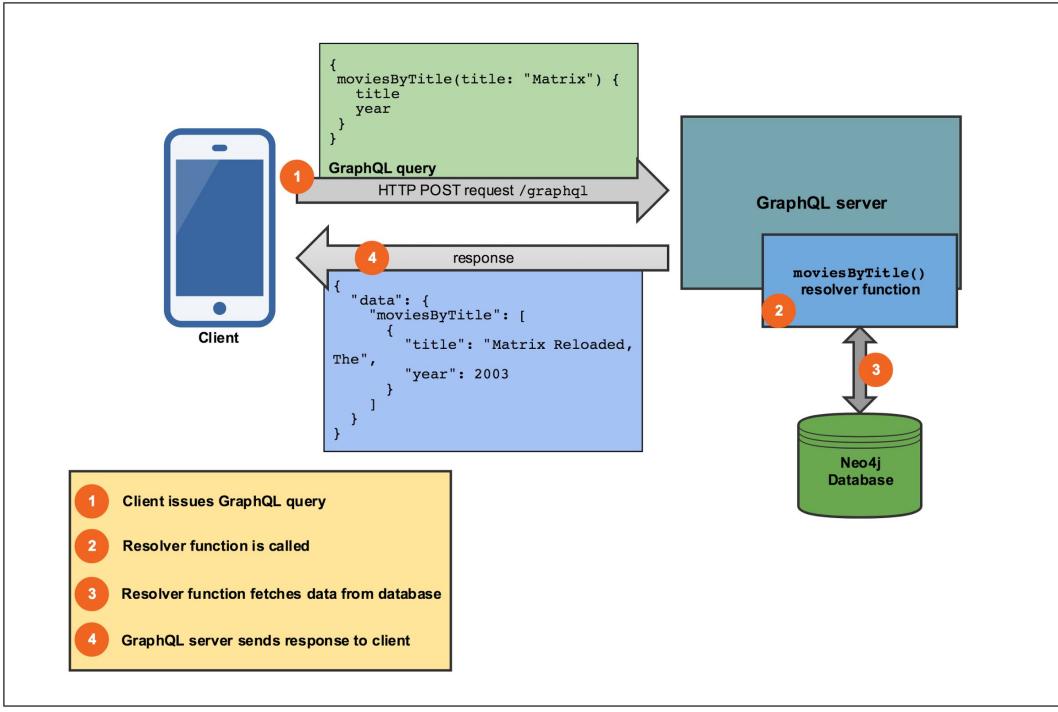
similar: [Movie]

bit.ly/_graphiql

Building A GraphQL Service

- 1) Define a schema
- 2) Implement resolver functions
 - o Fetch data from data layer

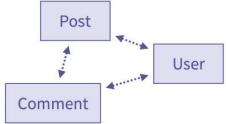
Building A GraphQL Service



<https://dzone.com/refcardz/an-overview-of-graphql>

GraphQL First Development

GraphQL First Development



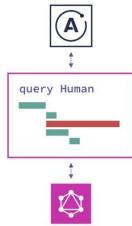
1. Design API schema

Contract between frontend and backend with a shared schema language



2. Build UI and backend

Parallelize with mocking, develop component-based UIs with GraphQL containers



3. Run in production

Static queries make loading predictable, schema tells you which fields are being used

1. Design API schema
2. Build UI and backend
3. Deploy!

- Schema is your friend
- GraphQL Schema is the API spec
 - Allows for simultaneous frontend and backend development
 - Enables introspection
 - Build other tools (graphiql)

IDL Schema Syntax



```
type Movie {
    movieId: ID!
    title: String
    year: Int
    plot: String
    poster: String
    imdbRating: Float
    genres: [String]
    similar(first: Int=3, offset:Int=0): [Movie]
}
```

```
type Query {
    moviesByTitle(subString: String!, first: Int=3, offset: Int=0): [Movie]
}
```

GraphQL Resolver Functions



```
Query: {
  moviesByTitle: (root, args, context) => {
    let session = context.driver.session();
    let query = "MATCH (movie:Movie) WHERE movie.title CONTAINS $subString RETURN movie LIMIT $first;";
    return session.run(query, args)
      .then( result => { return result.records.map(record => { return record.get("movie").properties }) })
  },
},
```

GraphQL Resolver Functions



```
const resolvers = {
  Query: {
    moviesByTitle: (root, args, context) => {
      let session = context.driver.session();
      let query = `MATCH (movie:Movie) WHERE movie.title CONTAINS $subString RETURN movie LIMIT $first;`;
      return session.run(query, args)
        .then( result => { return result.records.map(record => { return record.get("movie").properties })})
    },
  },
  Movie: {
    genres: (movie, _, context) => {
      let session = context.driver.session();
      let params = {movieId: movie.movieId};
      let query = `
        MATCH(m:Movie)-[:IN_GENRE]->(g:Genre)
        WHERE m.movieId = $movieId
        RETURN g.name AS genre
      `;
      return session.run(query, params)
        .then( result => { return result.records.map(record => {return record.get("genre")})})
    },
    similar: (movie, _, context) => {
      let session = context.driver.session();
      let params = {movieId: movie.movieId};
      let query = `
        MATCH (m:Movie) WHERE m.movieId = $movieId
        MATCH (m)-[:IN_GENRE]->(g:Genre)<-[:IN_GENRE]-(movie:Movie)
        WITH m, movie, COUNT(*) AS genreOverlap
        MATCH (m)<-[:RATED]-(:User)-[:RATED]->(movie:Movie)
        WITH movie,genreOverlap, COUNT(*) AS userRatedScore
        RETURN movie ORDER BY (0.9 * genreOverlap) + (0.1 * userRatedScore) DESC LIMIT 3
      `;
      return session.run(query, params)
        .then( result => {return result.records.map(record => {return record.get("movie").properties})})
    }
  }
};
```

Apollo Launchpad



Launchpad | New | Save | Download | Fork | GRAND Stack workshop - end | johnymontana ▾

GraphQL ► Prettify | Save | Reset | Headers

```
// GRAND Stack workshop - end state
// The goal of this section of the workshop is to complete our GraphQL server
// We start with a
// Need to query our Neo4j Database to ensure that we're
// Welcome to Launchpad!
// Log in to edit and save pads, run queries in GraphQL on the right.
// graphql-tools combines a schema string with resolvers.
// Import { makeExecutableSchema } from 'graphql-tools';
// Import {v1 as neo4j} from 'neo4j-driver';
// Construct a schema, using GraphQL schema language
const typeDefs =
  type Movie = {
    movieId: ID!
    title: String
    year: Int
    plot: String
    poster: String
    imdbRating: Float
    genres: [String]
    similar(first: Int=3, offset: Int=0): [Movie]
  }
type Query {
  moviesByTitle(subString: String!, first: Int=3, offset: Int=0): [Movie]
}
;
// Provide resolver functions for your schema fields
const resolvers = {
  Query: {
    moviesByTitle: (root, args, context) => {
      let session = context.driver.session();
      let query = "MATCH (movie:Movie) WHERE movie.title CONTAINS $subString RETURN movie LIMIT $first";
      return session.run(query, args)
        .then( result => { return result.records.map(record => { return record.get("movie").properties }) })
    },
    Movie: {
      genres: (movie, _, context) => {
        let session = context.driver.session();
        let params = {movieId: movie.movieId};
        let query = `
          MATCH(m:Movie)-[r{IN_GENRE}]->(g:Genre)
          WHERE m.movieId = $movieId
          RETURN g.name AS genre
        `;
        return session.run(query, params)
          .then( result => { return result.records.map(record => {return record.get("genre")})) })
      },
    }
  }
}
```

Welcome to GraphQL

```
query MovieListQuery($title: String!) {
  moviesByTitle(subString: $title, first: 30) {
    title
    movieId
    imdbRating
    plot
    poster
    year
    genres
    similar {
      movieId
      poster
      title
    }
  }
}
```

data:

```
{
  "data": {
    "moviesByTitle": [
      {
        "title": "Matrix, The",
        "movieId": "2571",
        "imdbRating": 8.7,
        "plot": "A computer hacker learns from mysterious rebels about the true nature of his reality and his role in the war against its controllers.",
        "poster": "http://ia.media-imdb.com/images/MV5BMjtxNDYxOTA4M15BMl5BanBnXkFzTgwNTk0NzQxMTE@._V1.SX300.jpg",
        "year": 1999,
        "genres": [
          "Thriller",
          "Sci-Fi",
          "Action"
        ],
        "similar": [
          {
            "movieId": "260",
            "poster": "http://ia.media-imdb.com/images/MV5B0TjyMDY2NGQtOGJjN100OTk4LWFhMdgtYmE3M2NiYzMoYTvnKkEyKfqcGdeQxVnTU1NTcwOTk@._V1.SX300.jpg",
            "title": "Star Wars: Episode IV - A New Hope"
          },
          {
            "movieId": "1196",
            "poster": "http://ia.media-imdb.com/images/MV5BmjZMzQmTgxN15BMl5BanBnXkFzTcwMDQzNjk2QQ@._V1.SX300.jpg",
            "title": "Star Wars: Episode V - The Empire Strikes Back"
          },
          {
            "movieId": "480",
            "poster": "http://ia.media-imdb.com/images/MV5BMjM2M0g0Mg0N15BMl5BanBnXkFzTgwNTM20TM5NDE@._V1.SX300.jpg",
            "title": "Jurassic Park"
          }
        ],
        "plot": "Neo and the rebel leaders estimate that they have 72 hours until 250,000 probes discover Zion and destroy it and its inhabitants. During this, Neo must decide how he can save Trinity from a dark fate in his dreams."
      }
    ]
  }
}
```

QUERY VARIABLES

```
1 {
  2   "title": "Matrix"
  3 }
```

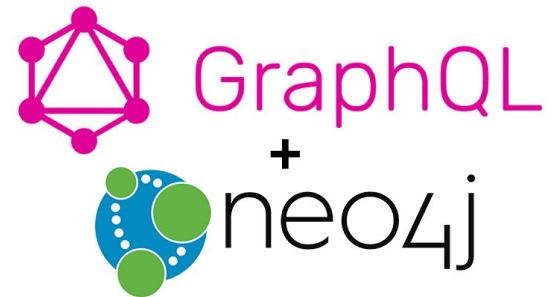
GraphQL Endpoint: <https://x57134qwl.lpqql.zone/graphql>

Secrets | npm Deps | Logs | Docs | Powered by Auth0 Extend

<https://launchpad.graphql.com/x57134qwl>

A GraphQL - Neo4j Integration?

- Developer productivity
- Translate GraphQL → Cypher?
- Improve performance?
- Expose Cypher through GraphQL?



neo4j-graphql



Neo4j-GraphQL Extension

Note This branch is for supporting Neo4j 3.2.

build passing

This implementation provides a GraphQL API to Neo4j, it comes as library but can also be installed as Neo4j server extension to act as a GraphQL endpoint. It turns GraphQL queries and mutations into Cypher statements and executes them on Neo4j.

We want to explore three approaches:

1. read schema / metadata from the database provide GraphQL DataFetcher that generate and run Cypher (WIP) ✓
2. make the same work with externally configured schema information (using IDL) ✓

github.com/neo4j-graphql/neo4j-graphql

circled | passing

neo4j-graphql-js

A GraphQL to Cypher query execution layer for Neo4j and JavaScript GraphQL implementations.

neo4j-graphql-js is in early development. There are rough edges and APIs may change. Please file issues for any bugs that you find or feature requests.

Installation and usage

Install

```
npm install --save neo4j-graphql-js
```

Then call `neo4jgraphql()` in your GraphQL resolver. Your GraphQL query will be translated to Cypher and the query passed to Neo4j.

```
import {neo4jgraphql} from 'neo4j-graphql-js';

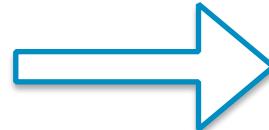
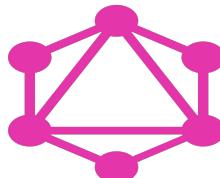
const resolvers = {
  Query: {
    Movie(object, params, ctx, resolveInfo) {
      return neo4jgraphql(object, params, ctx, resolveInfo);
    }
  }
};
```

github.com/neo4j-graphql/neo4j-graphql-js

neo4j.com/developer/graphql/

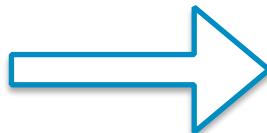
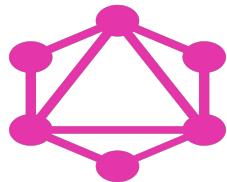
Use The Schema, Luke

```
1  type Movie {  
2      movieId: ID!  
3      title: String  
4      year: Int  
5      plot: String  
6      poster: String  
7      imdbRating: Float  
8      genres: [String]  
9      similar(first: Int = 3, offset: Int = 0): [Movie] @cypher(statement: "WITH {this} AS this MATCH (this)--(:Genre)--(o:Movie) RETURN o")  
10     mostSimilar: Movie @cypher(statement: "WITH {this} AS this RETURN this")  
11     degree: Int @cypher(statement: "WITH {this} AS this RETURN SIZE((this)--())")  
12     actors(first: Int = 3, offset: Int = 0): [Actor] @relation(name: "ACTED_IN", direction:"IN")  
13     avgStars: Float  
14     filmedIn: State @relation(name: "FILMED_IN", direction: "OUT")  
15  }  
16  
17  type State {  
18      name: String  
19  }  
20  
21  type Actor {  
22      id: ID!  
23      name: String  
24      movies: [Movie]  
25  }  
26  
27  type User {  
28      id: ID!  
29      name: String  
30  }  
31  
32  type Query {  
33      Movie(id: ID, title: String, year: Int, plot: String, poster: String, imdbRating: Float, first: Int, offset: Int): [Movie]  
34      MoviesByYear(year: Int): [Movie]  
35      AllMovies: [Movie]  
36      MovieById(movieId: ID!): Movie  
37  }
```



openCypher

Use The Schema, Luke



openCypher

```
{  
  Movie(title: "River Runs Through It, A") {  
    title  
    year  
    imdbRating  
  }  
}
```

```
MATCH (movie:Movie {title:"River Runs Through It, A"})  
RETURN movie { .title , .year , .imdbRating } AS movie  
SKIP 0
```

<https://github.com/neo4j-graphql/>

Improved Performance

```
{  
  Movie(title: "River Runs Through It, A") {  
    title  
    year  
    imdbRating  
    actors {  
      name  
    }  
  }  
}
```



```
MATCH (movie:Movie {title:"River Runs Through It, A"})  
RETURN movie { .title , .year , .imdbRating,  
  actors: [(movie)<-[ACTED_IN]-(movie_actors:Actor) | movie_actors { .name } ] }  
AS movie  
SKIP 0
```

openCypher

- N+1 query problem
 - Batching
- GraphQL → single Cypher query
 - Single round trip to database

Expose Cypher in GraphQL

- GraphQL directives
- **@cypher** schema directive
 - Map GraphQL fields to a Cypher query

```
type Movie {  
    movieId: ID!  
    title: String  
    year: Int  
    plot: String  
    poster: String  
    imdbRating: Float  
    genres: [String]  
    similar(first: Int = 3, offset: Int = 0): [Movie] @cypher(statement: "MATCH (this)--(:Genre)--(o:Movie) RETURN o")  
    degree: Int @cypher(statement: "RETURN SIZE((this)--())")  
    actors(first: Int = 3, offset: Int = 0): [Actor] @relation(name: "ACTED_IN", direction:"IN")  
    avgStars: Float  
    filmedIn: State @relation(name: "FILMED_IN", direction: "OUT")  
}
```

@cypher Schema Directives

```
{  
  Movie(title: "River Runs Through It, A") {  
    title  
    year  
    imdbRating  
    actors {  
      name  
    }  
    similar(first: 3) {  
      title  
    }  
  }  
}
```



```
MATCH (movie:Movie {title:"River Runs Through It, A"})  
RETURN movie { .title , .year , .imdbRating,  
actors: [(movie)<-[ACTED_IN]-(movie_actors:Actor) | movie_actors { .name }],  
similar: [ x IN apoc.cypher.runFirstColumn()  
WITH {this} AS this  
MATCH (this)-[:IN_GENRE]->(:Genre)<-[:IN_GENRE]-(o:Movie)  
RETURN o",  
{this: movie}, true) | x { .title }][..3]  
} AS movie  
SKIP 0
```

openCypher

- Still a single Cypher query / single round-trip
 - **@cypher** annotated query becomes a sub-query

neo4j-graphql-js



Auto-generate Cypher queries

Works with apollo-server, graphql-tools, graphql-js,...

A Launchpad New Save Download Fork neoj-graphql-jp movies example

① johnmynonta -

```
// Welcome to Launchpad!
// Log in to edit and save pads, run queries in GraphQL on the right.
// Click "Download" above to get a zip with a standalone Node.js server.
// See docs and examples at https://github.com/apollolgraph/graphql-awesome-launchpad

// graphql-tools combines a schema string with resolvers.
// Import `makeExecutableSchema` from `graphql-tools`;
// Import `(v4 as needed)` from `neoj`-driver`;
// Import `(neojgraphql)` from `neojgraphql/jp`;
// Import `__schema` from `neojgraphql/jp`;

const typeDefs = `
  type Movie {
    id: ID!
    title: String
    year: Int
    plot: String
    poster: String
    imdbRating: Float
    genres: [String]
    similar(first: Int = 3, offset: Int = 0): [Movie] @cypher(statement: "MATCH (this)-->(:Genre)-->(:Movie) RETURN mostSimilar: Movie @cypher(statement: "WITH this AS this RETURN this")")
    degree: Int @cypher(statement: "WITH this AS this RETURN SIZE((this)--())")
    actors(first: Int = 3, offset: Int = 0): [Actor] @relation(name: "ACTED_IN", direction: "IN")
    avgStars: Float
  }

  interface Person {
    id: ID!
    name: String
  }

  type Actor implements Person {
    id: ID!
    name: String
    movies: [Movie]
  }

  type User implements Person {
    id: ID!
    name: String
  }

  type Query {
    movie(id: ID!, title: String, year: Int, plot: String, poster: String, imdbRating: Float, first: Int, offset: Int): Movie
  }
};

const resolvers = {
  // root entry point to GraphQL service
  Query: {
    fetchMoviesByTitleSubstr: async (Movie, {param, ctx, resolveInfo}) => {
      // neojgraphql inspects the GraphQL query and schema to generate a single Cypher query.
      // ...
    }
  }
};

// GraphQL Endpoint: https://7kpp8pdj-lp gql.zone/graphql
```

GRAPHQL ▶ Prettyprint

```
1 {
  2   Movie(first:3) {
  3     title
  4     similar(first: 3) {
  5       title
  6     }
  7     actors {
  8       name
  9     }
 10   }
 11 }
12 }
```

Saved Reset Headers

```
{
  "data": {
    "Movie": [
      {
        "title": "Toy Story",
        "similar": [
          {
            "title": "Boxtrolls, The"
          },
          {
            "title": "The Book of Life"
          },
          {
            "title": "Teenage Mutant Ninja Turtles"
          }
        ],
        "actors": [
          {
            "name": "Tom Hanks"
          },
          {
            "name": "Jim Carrey"
          },
          {
            "name": "Don Rickles"
          },
          {
            "name": "Tim Allen"
          }
        ],
        "titles": [
          {
            "title": "Jumanji",
            "similar": [
              {
                "title": "Boxtrolls, The"
              },
              {
                "title": "The Book of Life"
              },
              {
                "title": "Teenage Mutant Ninja Turtles"
              }
            ],
            "actors": [
              {
                "name": "Robin Williams"
              },
              {
                "name": "Bradley Pierce"
              },
              {
                "name": "Kirsten Dunst"
              }
            ]
          }
        ]
      }
    ]
  }
}
```

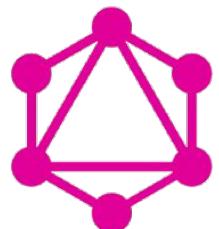
QUERY VARIABLES

DOCs Powered by AutoHotExt

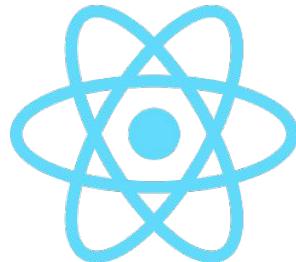
<https://launchpad.graphql.com/7kp8l0p4j>

github.com/neo4j-graphql/neo4j-graphql-js

The GRAND stack



GraphQL



React



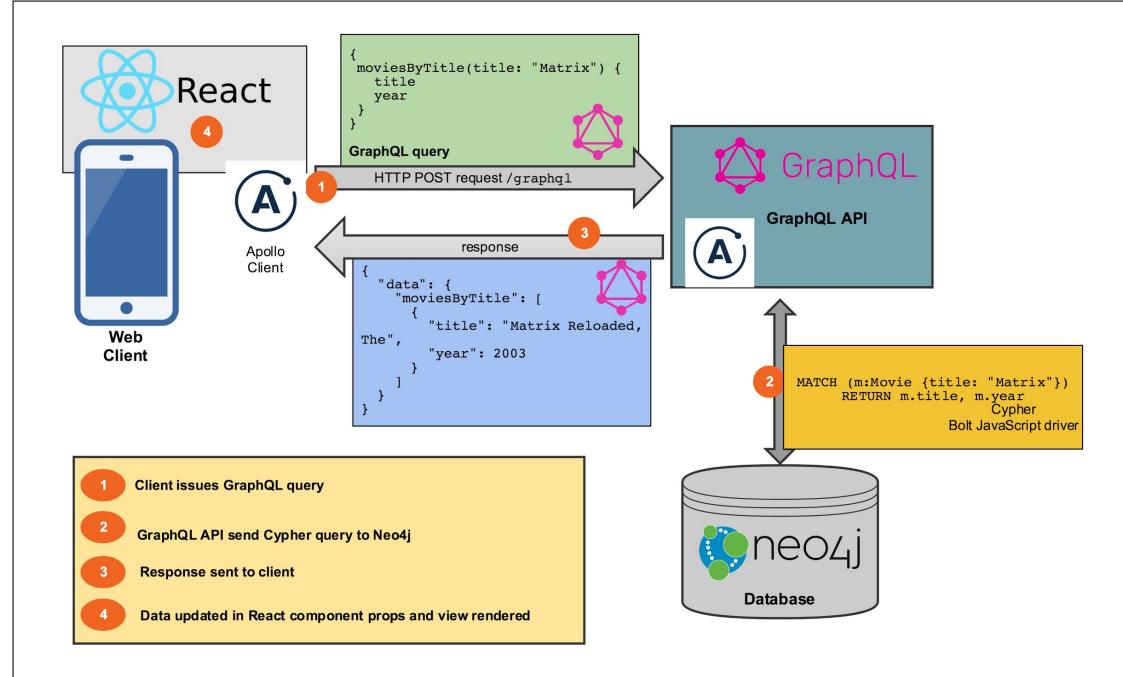
The GRAND stack

GraphQL

React

APOLLO

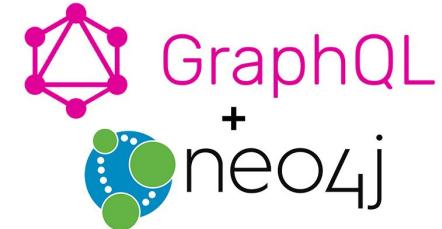
neo4j



Looking Forward

- Active development
 - Feedback driven :-)
- Features
 - Subscriptions
 - Authentication / Authorization
 - ????

will@neo4j.com



neo4j.com/developer/graphql/

GRANDSTACK

grandstack.io/

(you)-[:HAVE]->(questions)<-[:ANSWERS]-(will)