Full Stack Development With

GraphQL + Neo4j

William Lyon
@lyonwj
lyonwj.com

GraphConnect
Powered by Neo4j

William Lyon

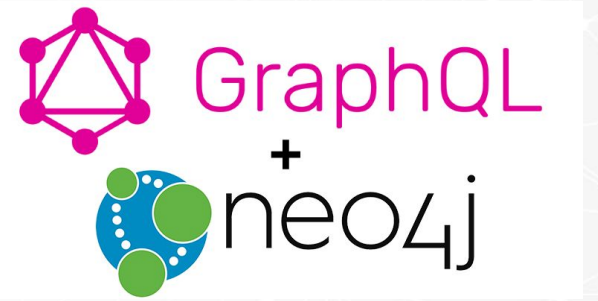
DevRel Engineering @neo4j

will@neo4j.com
@lyonwj
lyonwj.com

neo4j

GraphQL + Neo4j

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



An Overview Of GraphQL

- A new paradigm for buildings 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

GraphiQL ▶ Prettify History

```
query MovieSearch($title: String!, $limit: Int!)
{
  movies(subString:$title, limit:$limit) {
    movieId
    title
    year
    genres
    poster
    plot
    imdbRating
    similar {
      title
      poster
      year
    }
  }
}
```

QUERY VARIABLES

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


```
{
  "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 **Movie** ✕

Search Movie...

No Description

FIELDS

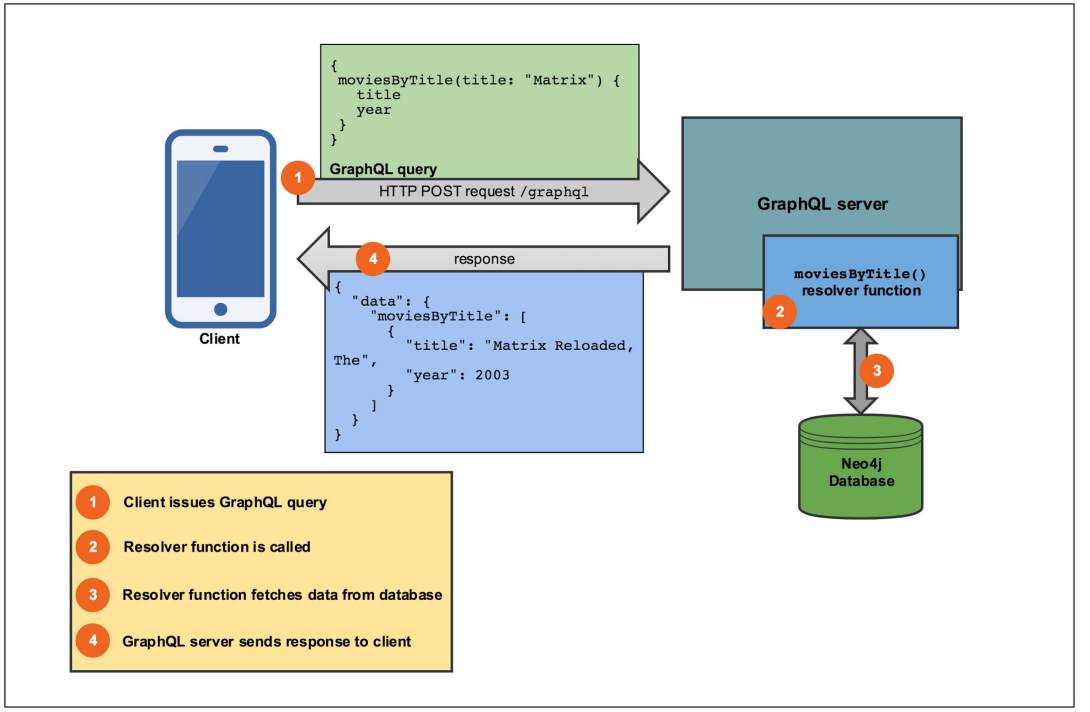
movieId: String!
title: String
year: Int
plot: String
poster: String
imdbRating: Float
genres: [String]
similar: [Movie]

bit.ly/_graphiql

Building A GraphQL Service

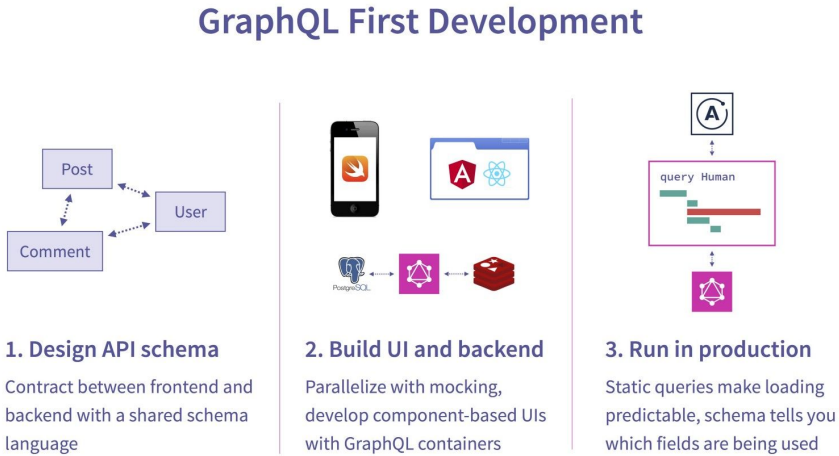
1) Define a schema
2) Implement resolver functions
    - Fetch data from data layer

Building A GraphQL Service



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

GraphQL First Development



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)

https://dev-blog.apollodata.com/navigating-your-transition-to-graphql-28a4dfa3acfb

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 })})
  },
},
```

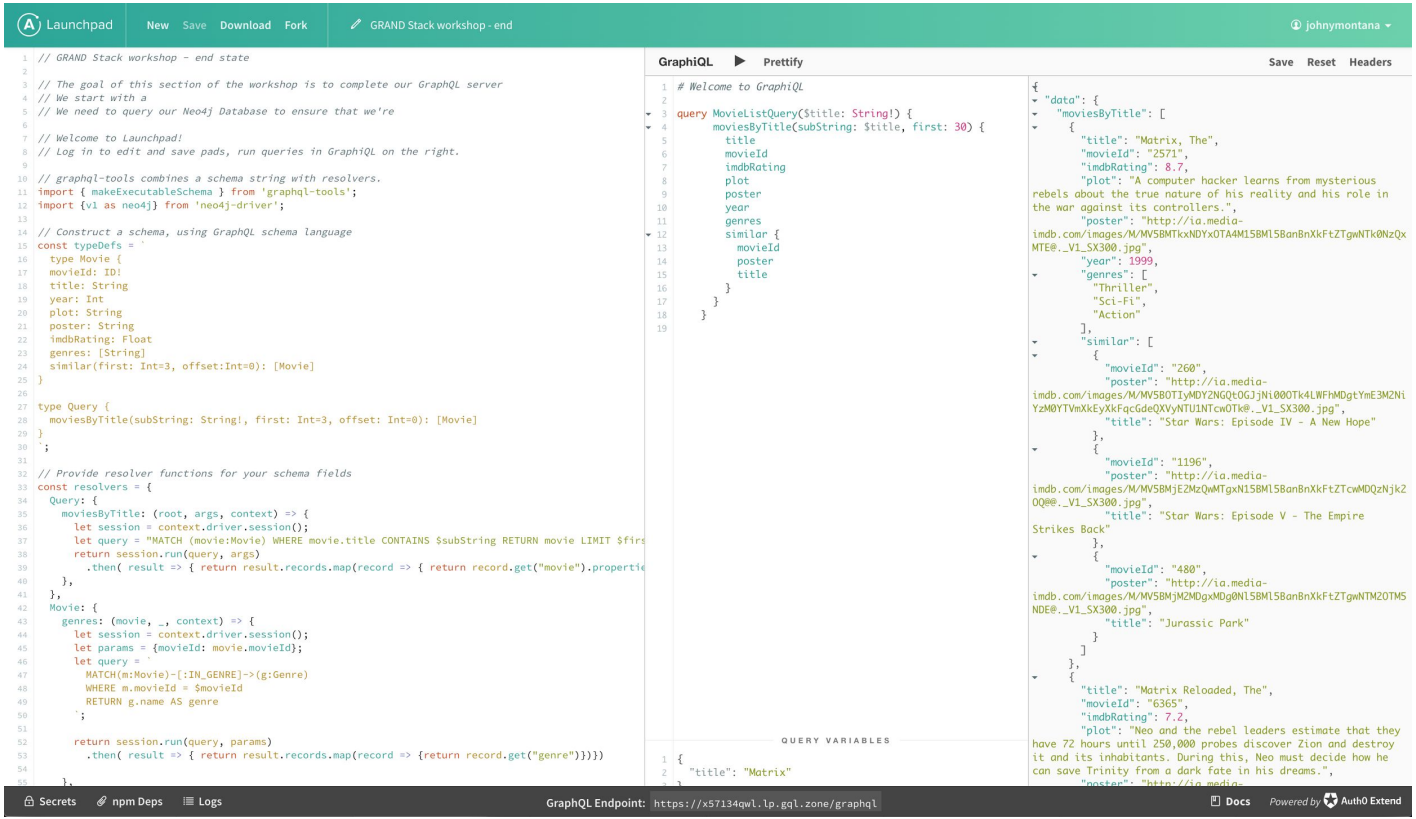
https://github.com/grand-stack/grand-stack-movies-workshop/blob/master/graphql/end/src/schema.js

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})})
    }
  }

};
```

https://github.com/grand-stack/grand-stack-movies-workshop/blob/master/graphql/end/src/schema.js

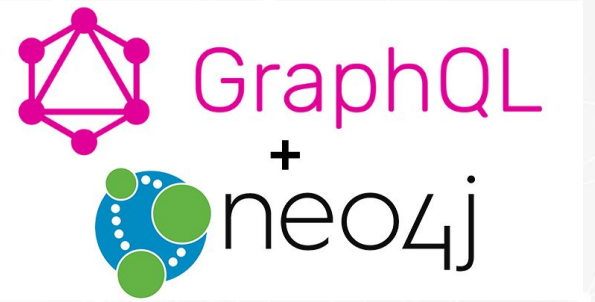
Apollo Launchpad



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

circleci 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.

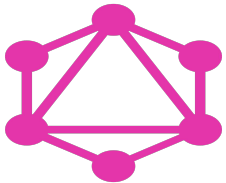
```
import {neo4jgraphql} from 'neo4j-graphql-js';

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

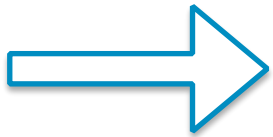
github.com/neo4j-graphql/neo4j-graphql-js

neo4j.com/developer/graphql/

Use The Schema, Luke

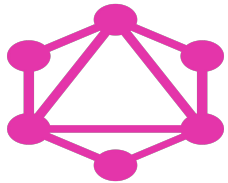
```graphql
type Movie {
    movieId: ID!
    title: String
    year: Int
    plot: String
    poster: String
    imdbRating: Float
    genres: [String]
    similar(first: Int = 3, offset: Int = 0): [Movie] @cypher(statement: "WITH {this} AS this MATCH (this)--(:Genre)--(o:Movie) RETURN o")
    mostSimilar: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

type State {
    name: String
}

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

type User {
    id: ID!
    name: String
}

type Query {
    Movie(id: ID, title: String, year: Int, plot: String, poster: String, imdbRating: Float, first: Int, offset: Int): [Movie]
    MoviesByYear(year: Int): [Movie]
    AllMovies: [Movie]
    MovieById(movieId: ID!): Movie
}
```





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
    }
  }
}
```

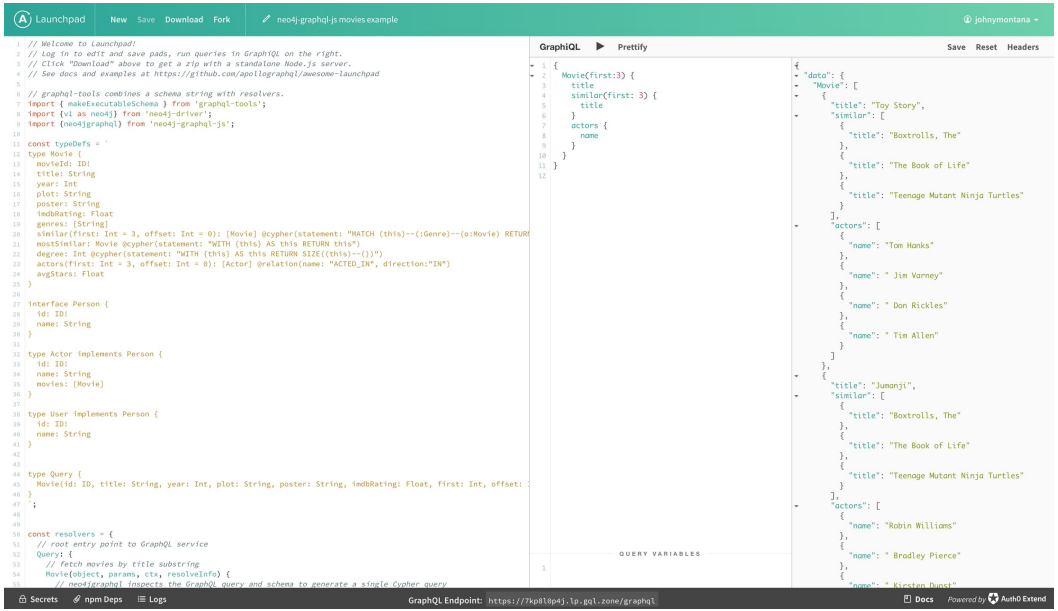
```
MATCH (movie:Movie {title:"River Runs Through It, A"})
RETURN movie { .title , .year , .imdbRating,
  actors: [(movie)<-[ACTED_IN]-(movie_actors:Actor) | movie_actors { .name }],
  similar: [ x IN apoc.cypher.runFirstColumn("
        WITH {this} AS this
        MATCH (this)-[:IN_GENRE]->(:Genre)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

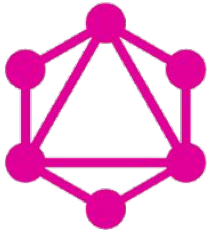
Auto-generate Cypher queries
Works with apollo-server, graphql-tools, graphql-js,...



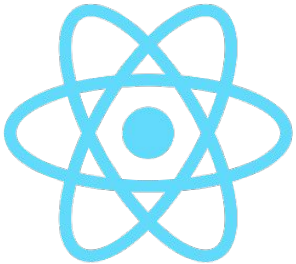
https://launchpad.graphql.com/7kp8l0p4j

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

The GRAND stack



GraphQL



React

grandstack.io

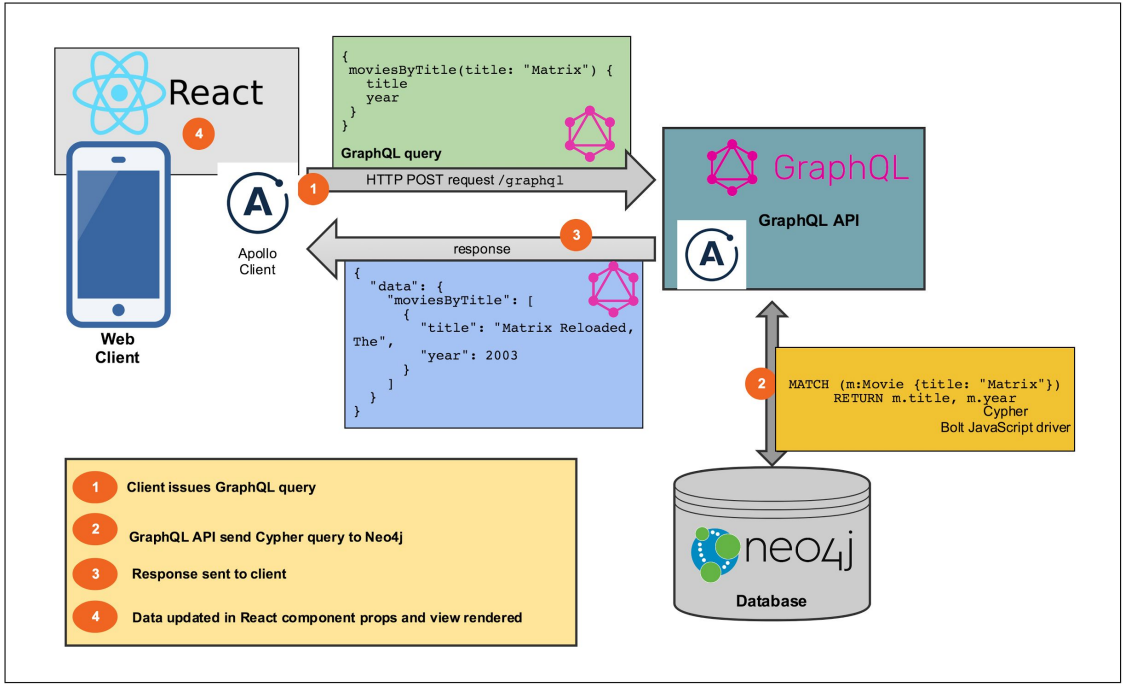
The GRAND stack

GraphQL

React

APOLLO

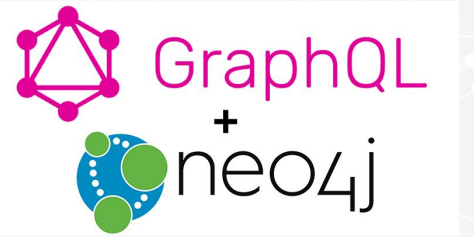
neo4j



grandstack.io

Looking Forward

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



neo4j.com/developer/graphql/

GRANDSTACK

grandstack.io/

will@neo4j.com

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