DEMYSTIFYING GRAPHQL CLIENTS

@chrisbiscardi

@chrisbiscardi

Design Systems and UI Platform @Dropbox

GRAPHQL

- What is it?
- Raw Queries
- Why Use a Client?
- Apollo 1.0
- Relay Modern

What is GraphQL?

A query language for your API

Describe your Data

```
type Project {
  name: String
  tagline: String
  contributors: [User]
}
```

Ask for What You Want

```
{
  project(name: "GraphQL") {
    tagline
  }
}
```

Get Predictable Results

```
{
  "project": {
    "tagline": "A query language for APIs"
  }
}
```

QUERYING GRAPHQL

- Fields
- Arguments
- Aliases
- Fragments
- Variables
- Operation Names
- Directives
- Mutations
- Meta Fields
- Errors
- Pagination
- Request Form
- Response Form
- Caching

FIELDS

```
{
  hero {
    name
    friends {
      name
    }
  }
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "friends": [
        {
          "name": "Luke Skywalker"
        },
        {
          "name": "Han Solo"
        },
        {
          "name": "Leia Organa"
        }
      ]
    }
  }
}
```

ARGUMENTS

```
{
  human(id: "1000") {
    name
    height(unit: FOOT)
  }
}
```

```
{
  "data": {
    "human": {
      "name": "Luke Skywalker",
      "height": 5.6430448
    }
  }
}
```

FRAGMENTS

```
{
  leftComparison: hero(episode: EMPIRE) {
    ...comparisonFields
  }
  rightComparison: hero(episode: JEDI) {
    ...comparisonFields
  }
}

fragment comparisonFields on Character {
  name
  appearsIn
  friends {
    name
  }
}
```

```
{
  "data": {
    "leftComparison": {
      "name": "Luke Skywalker",
      "appearsIn": [
        "NEWHOPE",
        "EMPIRE",
        "JEDI"
      ],
      "friends": [
        {
          "name": "Han Solo"
        },
        {
          "name": "Leia Organa"
        },
        {
          "name": "C-3PO"
        },
        {
          "name": "R2-D2"
        }
      ]
    },
    "rightComparison": {
      "name": "R2-D2",
      "appearsIn": [
        "NEWHOPE",
        "EMPIRE",
        "JEDI"
      ],
      "friends": [
        {
          "name": "Luke Skywalker"
        },
        {
          "name": "Han Solo"
        },
        {
          "name": "Leia Organa"
        }
      ]
    }
  }
}
```

VARIABLES

```
{
  query: `query HeroNameAndFriends($episode: Episode) {
    hero(episode: $episode) {
      name
      friends {
        name
      }
    }
  }`,
  variables: {
    "episode": "JEDI"
  }
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "friends": [
        {
          "name": "Luke Skywalker"
        },
        {
          "name": "Han Solo"
        },
        {
          "name": "Leia Organa"
        }
      ]
    }
  }
}
```

OPERATION NAMES

```
query HeroNameAndFriends {
    hero {
      name
      friends {
        name
      }
    }
  }
```

DIRECTIVES

```
{
  query: `query Hero($episode: Episode, $withFriends: Boolean
  hero(episode: $episode) {
      name
      friends @include(if: $withFriends) {
        name
      }
    }
  }`,
    variables: {
    "episode": "JEDI",
    "withFriends": false
  }
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2"
    }
  }
}
```

MUTATIONS

```
{
  query: `mutation CreateReviewForEpisode($ep: Episode!, $rev
    createReview(episode: $ep, review: $review) {
      stars
      commentary
    }
  }`,
    variables: {
    "ep": "JEDI",
    "review": {
      "stars": 5,
      "commentary": "This is a great movie!"
    }
  }
}
```

```
{
  "data": {
    "createReview": {
      "stars": 5,
      "commentary": "This is a great movie!"
    }
  }
}
```

ERRORS

```
# INVALID: primaryFunction does not exist on Character
{
  hero {
    name
    primaryFunction
  }
}
```

```
{
  "errors": [
    {
      "message": "Cannot query field "primaryFunction" on typ
      "locations": [
        {
          "line": 5,
          "column": 5
        }
      ]
    }
  ]
}
```

PAGINATION

CACHING

WHY USE A CLIENT?

> **Allow the application developer to easily execute GraphQL queries, and configure transport-specific features like headers.**

- Apollo DevBlog

Ensure that all GraphQL results currently being displayed in an app are consistent with one another.

- Apollo DevBlog

> **Provide flexible ways to update the cache with results from the server when using mutations, pagination, subscriptions, and more.**

- Apollo DevBlog

Apollo Client

> **The flexible, production ready GraphQL client for React and native apps.**

- Apollo Docs

WHAT'S IN APOLLO?

- Multi-Platform
- Queries
- Caching
- Mutations
- Optimistic UI
- Subscriptions
- Pagination
- Server Side
- Prefetching
- Network Layer
- Authentication

PROJECT GOALS

- Incrementally Adoptable
- Universally Compatible
- Simple Getting Started
- Inspectable and Understandable
- Build for Interactive Apps
- Small and Flexible
- Community Driven

INTERESTING TIDBITS

- Written in TypeScript
- Built on Redux
- fetch() polyfill

MODULARITY

- graphql-tag
- graphql-anywhere
- graphql-document-collector
- apollo-codegen
- persistgraphql
- eslint-plugin-graphql
- jest-transform-graphql

GRAPHQL-TAG

```
gql`
    query TodoApp {
      todos {
        id
        text
        completed
      }
    }
  `
```

GRAPHQL-ANYWHERE

```
const query = gql`
  {
    div {
      s1: span(id: "my-id") {
        text(value: "This is text")
      }
      s2: span
    }
  }
`;

assert.equal(
  renderToStaticMarkup(gqlToReact(query)),
  '<div><span id="my-id">This is text</span><span></span></div>'
);
```

GRAPHQL-DOCUMENT-COLLECTOR

```
graphql-document-collector '**/*.graphql' > documents.json
```

APOLLO-CODEGEN

```ts
//  This file was automatically generated and should not be edited.

// The episodes in the Star Wars trilogy
export type Episode =
  "NEWHOPE" | // Star Wars Episode IV: A New Hope, released in 1977.
  "EMPIRE" | // Star Wars Episode V: The Empire Strikes Back, released in 1980.
  "JEDI"; // Star Wars Episode VI: Return of the Jedi, released in 1983.


export interface HeroNameQueryVariables {
  episode: Episode | null;
}

export interface HeroNameQuery {
  hero: DescribeHeroFragment;
}

export interface DescribeHeroFragment {
  name: string;
  appearsIn: Array< Episode | null >;
}
```

PERSISTGRAPHQL

```
persistgraphql src/ extracted_queries.json

import queryMap from ‘../extracted_queries.json’;
import { invert } from 'lodash';
app.use(
  '/graphql',
  (req, resp, next) => {
    if (config.persistedQueries) {
      const invertedMap = invert(queryMap);
      req.body.query = invertedMap[req.body.id];
    }
    next();
  },
);
```

ESLINT-PLUGIN-GRAPHQL

```
// .eslintrc.js
module.exports = {
  parser: "babel-eslint",
  rules: {
    "graphql/template-strings": ['error', {
      env: 'apollo',
      schemaJson: require('./schema.json'),
    }]
  },
  plugins: [
    'graphql'
  ]
}
```

JEST-TRANSFORM-GRAPHQL

```
{
  "jest": {
    "transform": {
      "\.(gql|graphql)$": "jest-transform-graphql",
      ".*": "babel-jest"
    }
  }
}
```

SIMPLE QUERY

```
import React, { Component, PropTypes } from 'react';
import { gql, graphql } from 'react-apollo';

class Profile extends Component { ... }

const CurrentUserForLayout = gql`
        query CurrentUserForLayout {
          currentUser {
            login
            avatar_url
          }
        }
        `;

const ProfileWithData = graphql(CurrentUserForLayout)(Profile);
```

PROP-TYPES

```
Profile.propTypes = {
  data: PropTypes.shape({
    loading: PropTypes.bool.isRequired,
    currentUser: PropTypes.object,
  }).isRequired,
};
```

THERE'S A LOT HERE

```
query getUserAndLikes($id: ID!) {
  user(userId: $id) { name }
  likes(userId: $id) { count }
}
```

```
data: {
  user: { name: "James" },
  likes: { count: 10 },
  loading: false,
  error: null,
  variables: { id: 'asdf' },
  refetch() { ... },
  fetchMore() { ... },
  startPolling() { ... },
  stopPolling() { ... },
  // ... more methods
}
```

QUERY APIS

- props.data
- .loading/.error
- .networkStatus
- .variables
- .refetch
- .fetchMore
- .subscribeToMore
- .startPolling/stopPolling
- .updateQuery

DATA.NETWORKSTATUS

1. loading
2. setVariables
3. fetchMore
4. refetch
5. unused
6. poll
7. ready
8. error

MUTATIONS

```
import React, { Component, PropTypes } from 'react';
import { gql, graphql } from 'react-apollo';

const NewEntry = ({ submit }) => (
  <div onClick={() => submit('apollographql/apollo-client')}>
    Click me
  </div>
);

const submitRepository = gql`mutation submitRepository($repoFullName: String!) {
  submitRepository(repoFullName: $repoFullName) {
    createdAt
  }
}`;

const NewEntryWithData = graphql(submitRepository, {
  props: ({ mutate }) => ({
    submit: (repoFullName) => mutate({ variables: { repoFullName } }),
  }),
})(NewEntry);
```

OPTIMISTIC UI

```
const CommentPageWithData = graphql(submitComment, {
  props: ({ ownProps, mutate }) => ({
    submit: ({ repoFullName, commentContent }) => mutate({
      variables: { repoFullName, commentContent },

      optimisticResponse: {
        __typename: 'Mutation',
        submitComment: {
          __typename: 'Comment',
          postedBy: ownProps.currentUser,
          createdAt: +new Date,
          content: commentContent,
        },
      },
    });
  }),
})(CommentPage);
```

PREFETCHING

```
const FeedEntry = ({ entry, currentUser, onVote, client }) => {
  const repoLink = `/${entry.repository.full_name}`;
  const prefetchComments = (repoFullName) => () => {
    client.query({
      query: COMMENT_QUERY,
      variables: { repoName: repoFullName },
    });
  };

  return (
    <div className="media">
      ...
      <div className="media-body">
        <RepoInfo
          description={entry.repository.description}
          stargazers_count={entry.repository.stargazers_count}
          open_issues_count={entry.repository.open_issues_count}
          created_at={entry.createdAt}
          user_url={entry.postedBy.html_url}
          username={entry.postedBy.login}
        >
          <Link to={repoLink} onMouseOver={prefetchComments(entry.repository.full_name)}>
              View comments ({entry.commentCount})
          </Link>
        </RepoInfo>
      </div>
    </div>
  );
};

const FeedEntryWithApollo = withApollo(FeedEntry);
```

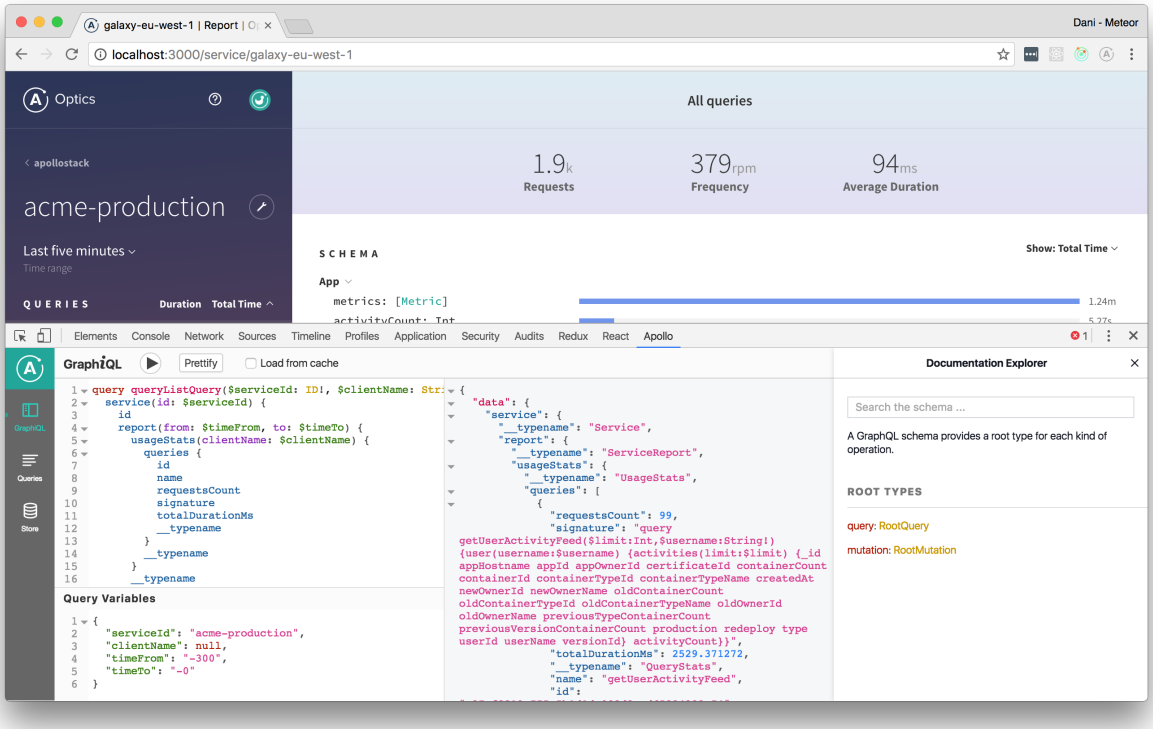
SSR

```
// server
import { getDataFromTree } from "react-apollo"

const client = new ApolloClient(....);

getDataFromTree(app).then(() => {
  const content = ReactDOM.renderToString(app);
  const initialState = {
    [client.reduxRootKey]: client.getInitialState()
  };

  const html = <Html content={content} state={initialState} />

  res.status(200);
  res.send(`<!doctype html>
${ReactDOM.renderToStaticMarkup(html)}`);
  res.end();
});
```

```
// client
const client = new ApolloClient({
  initialState: window.__APOLLO_STATE__,
});
```

SUBSCRIPTIONS

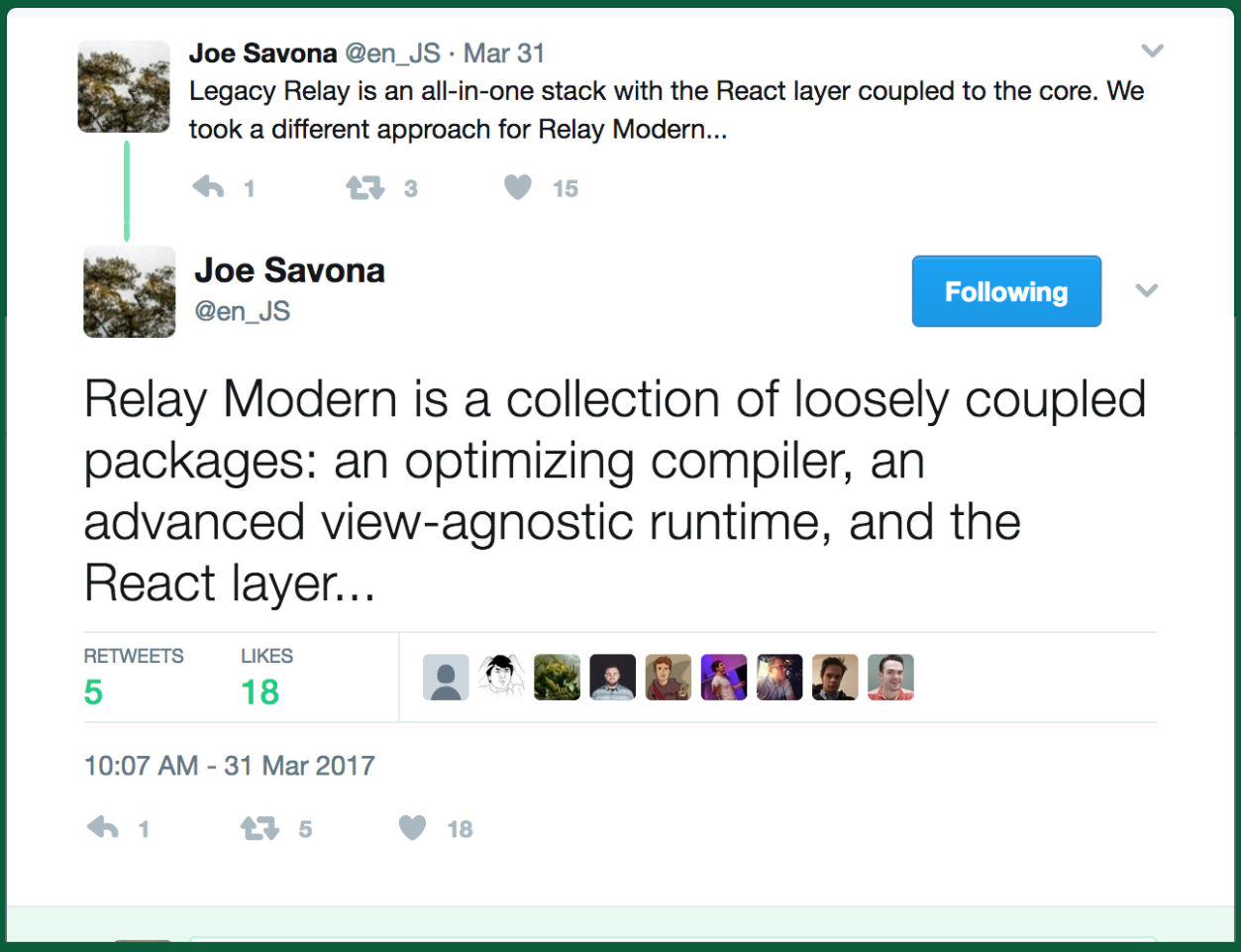
```
graphql(COMMENT_QUERY, {
  name: 'comments',
  options: ({ params }) => ({
    variables: {
      repoName: `${params.org}/${params.repoName}`
    },
  }),
  props: props => {
    return {
      subscribeToNewComments: params => {
        return props.comments.subscribeToMore({
          document: COMMENTS_SUBSCRIPTION,
          variables: {
            repoName: params.repoFullName,
          },
          updateQuery: (prev, {subscriptionData}) => {
            if (!subscriptionData.data) {
              return prev;
            }

            const newFeedItem = subscriptionData.data.commentAdded;

            return {
              ...prev,
              entry: {
                comments: [newFeedItem, ...prev.entry.activities]
              }
            };
          }
        });
      }
    };
  },
});
```


galaxy-eu-west-1 | Report
Dani - Meteor
localhost:3000/service/galaxy-eu-west-1
Optics
apollostack
acme-production
Last five minutes
Time range
QUERIES
Duration
Total Time
All queries
1.9k
Requests
379rpm
Frequency
94ms
Average Duration
SCHEMA
Show: Total Time
App
metrics: [Metric]
1.24m
Elements
Console
Network
Sources
Timeline
Profiles
Application
Security
Audits
Redux
React
Apollo
GraphiQL
Prettify
Load from cache
Documentation Explorer
Search the schema ...
A GraphQL schema provides a root type for each kind of operation.
ROOT TYPES
query: RootQuery
mutation: RootMutation
Query Variables

APOLLO DEVTOOLS

- GraphiQL
- Normalized Store Inspector
- Watched Query Inspector
- Redux DevTools

ʃ つ ◕_◕ ʅ つ Give RELAY Modern


Joe Savona @en_JS · Mar 31
Legacy Relay is an all-in-one stack with the React layer coupled to the core. We took a different approach for Relay Modern...
1
3
15
Joe Savona
@en_JS
Following
Relay Modern is a collection of loosely coupled packages: an optimizing compiler, an advanced view-agnostic runtime, and the React layer...
RETWEETS
5
LIKES
18
10:07 AM - 31 Mar 2017
1
5
18

RELAY MODERN

- Complete Rewrite
- Optimized for low-end mobile devices
- Simpler Developer Experience
- Client Schema Extensions
- Subscriptions
- Static Analysis
- Built in Flow Support
- Prototype "live" polling
- __generated__
- 20% Smaller than Classic

MODULARITY

- relay-compiler
- relay-runtime
- react-relay
- new babel-plugin-relay

RELAY-COMPILER

```
relay-compiler --src ./src --schema ./schema.graphql
```

```
import type {
  DictionaryComponent_word
} from './__generated__/DictionaryComponent_word.graphql';
```

RELAY MODERN

- Container
- Refetch Container
- Pagination Container
- Fragment Container
- Query Renderer
- Relay Environment

Pete Hunt
@floydophone

Following

Which GraphQL implementation have you had more success with?

34% Relay

66% Apollo

1,193 votes • Final results

RETWEETS 26 LIKES 59

2:34 AM - 13 Mar 2017

SIMILARITIES

- Modularity
- Static Analysis
- View/Data Colocation
- Network Layer Abstractions
- Subscriptions
- Client State

DIFFERENCES (RELAY VS APOLLO)

- Build Time vs Runtime AST
- Apollo's Fragment read/write APIs
- Custom Store vs Redux
- Dev Tools
- Ecosystem

GRAPHCOOL

Relay vs Apollo

https://www.graph.cool/docs/tutorials/relay-vs-apollo-iechu0shia/

GraphQL-Europe

urope's first GraphQL-dedicated conferenc

BERLIN 2017

Q&A

@chrisbiscardi