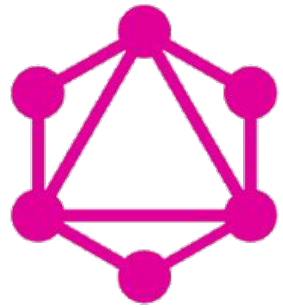
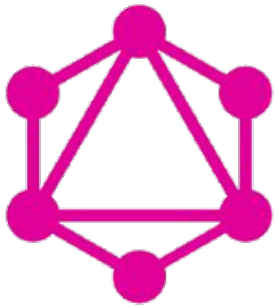


GraphQL

Recipes



GraphQL

Notes

This is not a beginner level talk on GraphQL

These notes are based on my personal experiences and YMMV

Perfectly OK to disagree with the content

What is GraphQL ?

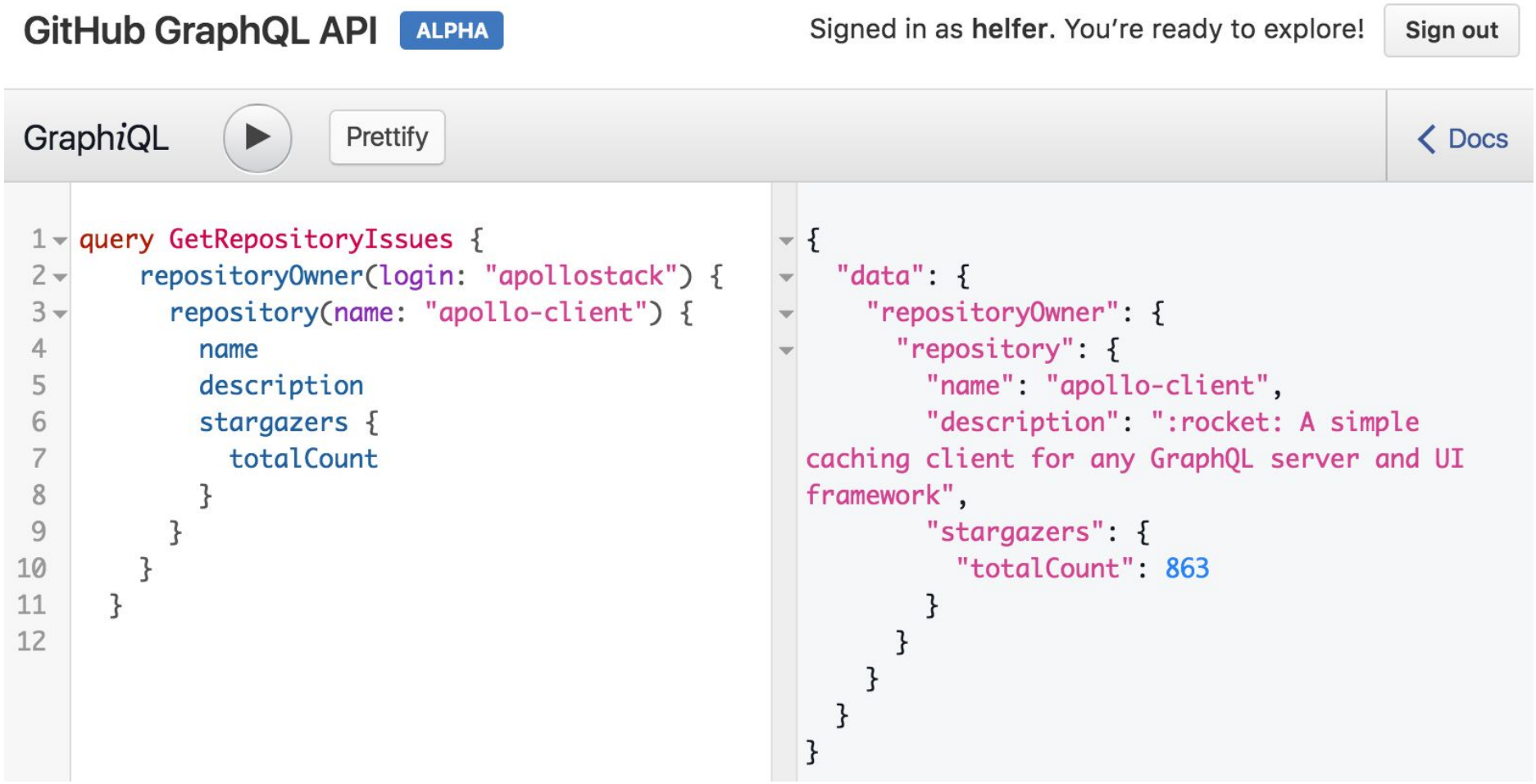
Birds eye view

SERVER

- Exposes an object graph called schema with types
- Exposes a set of operations called Query (read), Mutations (create, update, delete) and Subscriptions (web sockets)
- These operations are powered by functions called resolvers

CLIENT

- Consumes the schema by executing the exposed operations
- Requests only the data it needs as a result of these operation executions


GitHub GraphQL API
ALPHA
Signed in as helfer. You're ready to explore!
Sign out
GraphiQL
Prettify
Docs
query GetRepositoryIssues {
repositoryOwner(login: "apollostack") {
repository(name: "apollo-client") {
name
description
stargazers {
totalCount
}
}
}
}
{
"data": {
"repositoryOwner": {
"repository": {
"name": "apollo-client",
"description": ":rocket: A simple caching client for any GraphQL server and UI framework",
"stargazers": {
"totalCount": 863
}
}
}
}
}

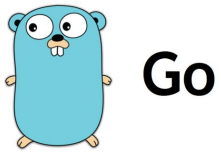
Why GraphQL ?

- Flexible, introspectable API putting API consumers first.
- Free input validation
- Solves what so many HyperMedia formats could not solve.
- Design by contract in a typesafe way.
- Plays well with other server technologies.
- Makes UI dev much easier

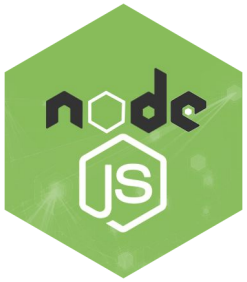
What tech stack should I use ??

- Personal preference
- Great community
- Productive
- Not heavily opinionated

- KGraphQL



- graphql-go



- Apollo Server
- TypeGraphql

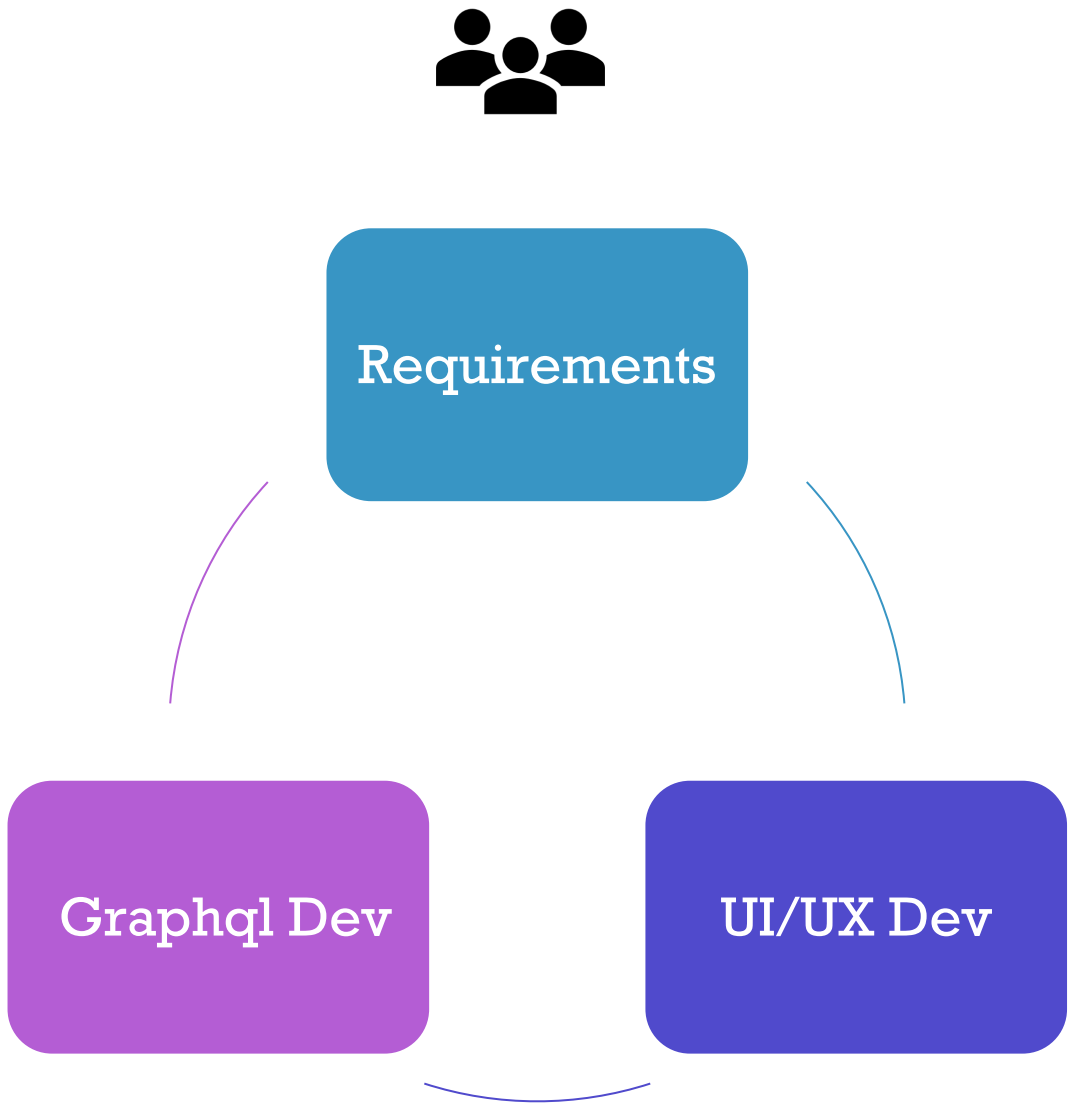
- graphql-ruby

- graphene

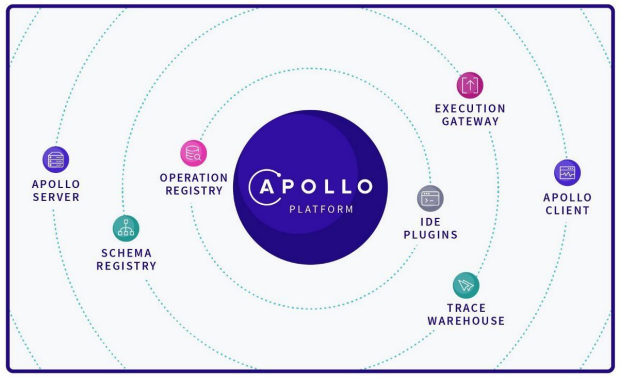
Find your own flavor of Graphql Server here https://graphql.org/code/

Lessons Learned

Be extremely consumer focused



AWS AppSync



GraphQL in the cloud ? (GrAAS)

Important stuff is still important

- Don’t need to rewrite everything in GraphQL
- Pick a pagination convention from the beginning and stick to it.
- Utilize GraphQL's ability to return errors. This is where explicit error codes are a good idea.
- Be very clear about the authentication and authorization strategy. Leverage an external service like Cognito/Auth0/Okta or a home grown solution as a service. Isolate this as much as you can.
- Must use log aggregation and monitoring to see what's going on with the service.
- Protect your API against over fetching. No one likes a slow API

Take care of the schema

- Use a code generator to generate language compatible types to keep up with the growing schema
- Use tools like graphql-inspector to ensure schema is always backwards compatible, type uniqueness and schema coverage. Helps control schema size as well.

```
Detected the following changes (6) between schemas:

✖ Field posts was removed from object type Query
✖ Field modifiedAt was removed from object type Post
✔ Field Post.id changed type from ID to ID!
✔ Deprecation reason on field Post.title has changed from No more used to
✔ Field Post.title changed type from String to String!
✔ Field Post.createdAt changed type from String to String!

error Detected 2 breaking changes
```

Schema Comparison

```
Detected 1 invalid document:

error in ./documents/post.graphql:

  - Cannot query field createdAtSomePoint on type Post. Did you mean createdAt?


Detected 1 document with deprecated fields:

warn in ./documents/post.graphql:

  - The field Post.title is deprecated. No more used
```

Schema to Document Validation

Schema Insights

```
type Post
●●●●● (96%) EmailPost
●●●●● (78%) BlogPost
●●●●● (72%) MailPost

type BlogPost
●●●●● (94%) MailPost
●●●●● (78%) Post
●●●●● (72%) EmailPost

type MailPost
●●●●● (94%) BlogPost
●●●●● (76%) EmailPost
●●●●● (72%) Post

type EmailPost
●●●●● (96%) Post
●●●●● (76%) MailPost
●●●●● (72%) BlogPost
```

```
success Schema coverage based on documents:

type Query {
  post x 1
  posts x 0
}

type Post {
  id x 1
  title x 1
  createdAt x 0
  modifiedAt x 0
}
```

Performance

- N+1 problem is very real. Use a library like dataloader that works for your stack
- Run performance tests against the hot queries and mutations to identify which ones.
- Since graphql resolvers can pull data from anywhere, don't hesitate to use high performance stacks or data source. Here is how Airbnb solved it.

Is there an ugly side to this GraphQL thing ?

- Metadata in the form of directives are not visible to the consumer of the API. Example auth scopes/constraints applied to schema
- Using GraphQL for uploading files feels very hacky.
- GraphQL subscriptions feel a bit half baked.
- Schema stitching, a technique where we combine schemas from other apis and delegate the operations to the respective schemas. It works great but can add work around authn/error logging/tracing.

Rahul Ballal

rballal@dius.com.au

@rahulballal7

GitHub

rahulballal