Introduction

Apollo is a GraphQL client/server for transporting data. While it doesn't yet have all the features that Meteor's pub/sub system has, it provides a way to get data from any database – not just MongoDB.

- Apollo docs

You can get started with Apollo and Meteor by creating a new Meteor application with the Apollo skeleton:

```
meteor create apollo-app --apollo
```

Apollo Client

Apollo client docs

Getting data

Instead of calling `Meteor.subscribe`, you will use queries to get data.

The main difference with subscriptions is that queries get called only once (by default) and don't get updated data like a subscription would. This is great for data that doesn't change often and where you don't need reactivity.

Changing data

Instead of calling a Meteor method with `Meteor.call`, you use a function called `mutate` to run a *mutator*, which is GraphQL's equivalent to a method.

Mutators are only run on the server, but they can return an object which then can update the local cache without the need to call a query again.

Apollo Server

Apollo server docs

Getting data

Instead of using `Meteor.publish` to define publications, you write resolve functions – called *resolvers* – that fetch different types of data in the query.

Changing data

Instead of using `Meteor.methods` to define methods, you write mutators – functions that *mutate* (change) data.

These are part of the resolver functions under `Mutation` key.

GraphQL

GraphQL is a query language for apps to get the data they want. Instead of the server deciding what's in a publication, the client uses GraphQL to say exactly which fields of which objects it wants.

- About GraphQL
- Intro to GraphQL
- GraphQL coming from REST

Advanced

Principled GraphQL

Latency

Meteor publications are blocking by default, whereas multiple GraphQL queries are executed in parallel. Publications stream data to the client as it arrives, whereas all the resolvers in a GraphQL query have to return before the data is sent to the client. (Although GraphQL is discussing adding the ability to stream results to the client as they come in.)

Meteor specific

Meteor has a specific Apollo package which includes user object into the context of a query.

```
meteor add apollo
```

On server you import `getUser` function and include it into the context option when setting up Apollo server:

```
import { ApolloServer } from 'apollo-server-express';
import { getUser } from 'meteor/apollo';
import typeDefs from '/imports/apollo/schema.graphql';
import { resolvers } from '/server/resolvers';

const server = new ApolloServer({
  typeDefs,
  resolvers,
  context: async ({ req }) => ({
    user: await getUser(req.headers.authorization)
  })
});
```

This will make user data available (if user is logged in) as the option in the query:

```
{
  Query: {
    userUniverses: async (obj, { hideOrgs }, { user }) => {
      if (!user) return null
      const selector = { userId: user._id, }
      if (hideOrgs) selector.organizationId = { $exists: false }
      return UniversesCollection.find(selector).fetch()
    }
  }
}
```

There are many other community packages that provide additional features or makes the initial setup easier, here is an incomplete list of some of them:

- quave:graphql - Utility package to create GraphQL setup in a standard way.
- cultofcoders:apollo - Meteor & Apollo integration.
- cultofcoders:graphql-loader - Easily load your GraphQL schema in your Meteor app!
- cultofcoders:apollo-accounts - Meteor accounts in GraphQL
- swydo:blaze-apollo - Blaze integration for the Apollo Client

- swydo:ddp-apollo - DDP link and server for Apollo.