…Introduction

Few months ago, I embarked on a full stack spatial data project at work. The project kicked off amazingly, until I was almost backed to the wall when I discovered that some of the data sources were served via a GraphQL API. Before now, I haven't worked with GraphQL. But, I have heard a lot about it and how amazing it is for querying data.

GraphQL is a query language for application programming interfaces (APIs) that prioritizes giving clients exactly the data they request. It's designed to make APIs flexible, fast and friendly. Basically, it is used to load data from a server to a client and it does this in a much more efficient manner than traditional methods and services.

I started off installing the GraphQL playground, which is now my go to tool to interact with GraphQL API and it offers a great workflow to understand the schema and structure of any GraphQL API. I recommend you try it out.

After hours of trying out the GraphQL playground, I finally understood the schema of the GraphQL API endpoint for the project I'm working on. Well, that was the easy part I must say.

Now that I can query the API, everything should seem great. Right? Well, it's not. Why? Well, the GraphQL API I was working with fetches the geospatial data as json file instead of geojson. Probably, because of my lack of understanding of how GraphQL truly worked with spatial data. The issue now is, I can see my data. But, just not in the format that I can work with. I am tempted to talk about the technicality behind it, but I won't… maybe another time.

How did I find out about ghql?

At work, our scripting language of choice is R. R is a free software environment for statistical computing and graphics. Well, that's how it's officially defined. But, trust me its gradually evolving to do more than statistical computing and making of beautiful graphics. It was time for me to programmatically access data with GraphQL API from R. I felt excited because, I'm an R lover. But, I knew it was a going to be a huge work, I guess I was ready for the huge work.

After spending days researching about the R packages that can interact with a GraphQL API, I found three packages. I picked `ghql` over the others because, it was an rOpenSci package.

Moving on

In order for R to interact with any GraphQL API, it requires a GraphQL client. That's where `ghql`, a GraphQL client for R, developed by Scott Chamberlain comes into play. Still confused? Well, so was I at first. I tried interacting with a GraphQL server in R and felt like giving up even before getting anywhere. Hopefully this blog post will assist others.

> GraphQL client and R connection flow

> GraphQL client and R connection flow

Working with `Countries List`, a GraphQL public API

Let's say you were working on a project that required country-specific data, such as currency, or language. You could get such data from the Countries GraphQL API which is a public GraphQL API for information about countries, continents, and languages. This public API uses Countries

List and provinces as data sources, so the schema follows the shape of those data, with a few exceptions:

- The codes used to key the objects in the original data are available as a code property on each item returned from the API.
- The country.continent and country.languages are objects and arrays of objects, respectively.
- Each Country has an array of states populated by their states/provinces, if any.

Loading the libraries

```
library(ghql)
library(jsonlite)
library(dplyr)
```

Link to the GraphQL schema api

```
link <- 'https://countries.trevorblades.com/'
```

Create a new graphqlClient object

```
conn <- GraphqlClient$new(url = link)
```

Define a Graphql Query

```
query <- '
query($code: ID!){
 country(code: $code){
 name
 native
 capital
 currency
 phone
 languages{
 code
 name
 }
 }
}'
```

The `ghql` query class and define query in a character string

```
new <- Query$new()$query('link', query)
```

Inspecting the schema

```
new$link
```

```
##
##
## query($code: ID!){
## country(code: $code){
## name
## native
## capital
```

```
## currency
## phone
## languages{
## code
## name
## }
## }
## }
```

Define a variable as a named list

```
variable <- list(
code = "DE"
)
```

Making a request, passing in the query and then the variables. Then you convert the raw object to a structured json object

```
result <- conn$exec(new$link, variables = variable) %>%
fromJSON(flatten = F)
result
```

```
## $data
## $data$country
## $data$country$name
## [1] "Germany"
##
## $data$country$native
## [1] "Deutschland"
##
## $data$country$capital
## [1] "Berlin"
##
## $data$country$currency
## [1] "EUR"
##
## $data$country$phone
## [1] "49"
##
## $data$country$languages
## code name
## 1 de German
```

Convert the json data into a tibble object

```
country_data <- result$data$country %>%
as_tibble()
country_data
```

```
## # A tibble: 1 x 6
## name native capital currency phone languages$code $name
##
## 1 Germany Deutschland Berlin EUR 49 de German
```

More examples

Working with a GraphQL API without a defined variable named list

```
link <- 'https://countries.trevorblades.com/'
# R6 class for constructing graphql queries
conn <- GraphqlClient$new(url = link)
## Define query
## Create a query class first
qry <- Query$new()
## The graphQL server schema
qry$query('x', '{
continent(code: "AF") {
 countries{
 code
 name
 native
 capital
 currency
 phone
 languages {
 name
 }
 }
}
 }
 ')
## Execute the query
res <- conn$exec(qry$queries$x)
# Convert the the output from raw to json format
res <- jsonlite::fromJSON(res,
flatten = TRUE)
## convert the from json to dataframe object
res_data <- res$data$continent$countries %>%
as_tibble()
## Inspect the first 6 rows of the data
res_data

## # A tibble: 58 x 7
## code name native capital currency phone languages
##
## 1 AO Angola Angola Luanda AOA 244
```

The last dance (conclusion)

So, you have stuck with me this far? Thanks!

My final thought. I think GraphQL can greatly simplify data needs for both client product developers, server-side engineers and data scientist. It’s still early to assertain the extent of it’s impact in the technological world. But, it seems very promising since the Team behind GraphQL are continously improving the technology, and there is a growing community.