Find Out Everything About a GitHub Repo With The GitHub GraphQL API And Go

Introduction

A lot of us are used to interacting with our github repos on the command line and in the GitHub web app. More advanced users can automate their interactions using the GitHub API's. GitHub has two API's: REST and GraphQL. Either one lets you access and automate processes. According to the documentation, the GraphQL API is targeted at more advanced usage.

This article is about using the GraphQL API to look at the details about any public repository, using Go.

REST API

From GitHub REST API Docs: "You can use GitHub's API to build scripts and applications that automate processes, integrate with GitHub, and extend GitHub. For example, you could use the API to triage issues, build an analytics dashboard, or manage releases."

GraphQL API

From GitHub GraphQL API Docs: "To create integrations, retrieve data, and automate your workflows, use the GitHub GraphQL API. The GitHub GraphQL API offers more precise and flexible queries than the GitHub REST API."

Differences

REST vs GraphQL

"Under REST architecture, data is returned to the client from the server in the whole-of-resource structure specified by the server."

"A data format describes how you would like the server to return the data, including objects and fields that match the server-side schema"

On other words, GraphQL servers and client can mix data from multiple resources and specify just what is needed, where REST sends a single 'document'.

Which One to Use

Get the scoop from the source:

Comparing GitHub's REST API and GraphQL API

In short, the GraphQL API allows fine grained access to its resources, where the REST API is less flexible and may give you more information than you might want. That's basically the difference between REST and GraphQL. HOWEVER, the easiest option to query the GraphQL API doesn't provide that level of granularity. More on that below.

Three ways to query the GraphQL API

Unlike the REST API, all access to the GitHub GraphQL API require authentication. Examples of that will be in the code.

1. Raw POST Requests

At the low level, a client queries a GraphQL API using an HTTP POST request. The payload is a GraphQL formatted structure that specifies what you want to get. This is dooable, but can be kind of klunky and hard coded. It's possible to handcraft the POST request payload, but it is a bit tricky to get everything right. So most (all?) users will use a GraphQL Client package to simplify the process. There is an example below.

2. Google go-github

Google has created a Go package that supports accessing the GitHub GraphQL API, at google/go-github. This is the easiest way to get at the GraphQL API, because it takes care of all the underlying GraphQL magic. It provides types and methods that correspond to the REST API.

This is the easiest way to go (pun), but the drawback is that is works like the REST API, returns whole documents rather than more fine grained requests that GraphQL is about.

3. A GraphQL client

For more fine grained access but with easier code than a raw POST, you can use a full client package. Here's a list of libraries for Go. Scroll down for clients. There are a couple of clients that have at least 1K GitHub stars. I chose shurcooL/graphql, pretty easy to use, sort of.

With a direct client, there is more work setting up types to match the requests. But it allows full up GraphQL queries that can drill down to exactly what you want.

GitHub GraphQL API Schema

Figuring out what you want to do is a bit daunting. The GitHub GraphQL API schema is massive, a bit over 1MB in size. You can download the schema. But the schema is sort of opaque to someone (like me) who isn't a GraphQL or GitHub API expert. You can grep it to find what you are looking for. A simpler way can be to use the GitHub GraphQL Explorer. Pretty easy to search, even if you only have a vague idea of what you are looking for. And it then gives you a definition of the type layout you need to set up.

In this case I first searched the schema for 'repositories' and found quite a few hits. It wasn't clear to me what to use. I went to the Explorer and it was much more friendly. It took me a bit of flailing but I found the 'search' query which is what you want to find specific things. "Perform a search across resources, returning a maximum of 1,000 results". In this case I wanted a list of public repositories from any owner.

The query looks like this, with the variable 'queryString'. 'org' is the name of the account owner you want to search. In this case I only wanted the repo names from the type 'Repository'. That type has a bunch of data you can add to it. Go into the explorer, search 'Docs' and look for 'Repository', no prefix or suffix. That gives you the entire type definition. In most cases you will use that type definition to create or use a corresponding Go struct type.

With this query I was able to get a list for any owner account. Substitute whichever owner in place of 'octocat' in the queryString. Add any of the fields defined in the Repository type. If you have a GitHub account, login and go to the Explorer and try this out.

```
query ($queryString: String!) {
  search(query: $queryString, type: REPOSITORY, first: 100) {
    repositoryCount
    edges {
      node {
        ... on Repository {
          name
        }
      }
    }
  }
}

{
"queryString": "owner:octocat"
}
```

The Example Application

The code of this example app builds a command line utility that lets you query for information about any public github repo. The app has examples of the three types of access that fetches a list of repositories for a specified org/user.

The app is called 'gh-repo', with 3 subcommands: raw,go-github,shurcool and one argument 'user'.

- $gh-schema raw [user]
- $gh-schema go-github [user]
- $gh-schema shurcool [user]

It does the access using your account based on the GITHUB_TOKEN environment variable. The app has code that does the authentication.

The API used in the example code is the top level "query search" which can be used to search for several different GitHub objects. In this case a search of repositories for a specified owner is performed. You can find its definition in the GraphQL Explorer. Open the Schema Docs (file icon in upper left of the dialog), click on "Query", then scroll down to "search", or use hourglass to find "search".

There are other ways to list particular things that give a similar result. You can use the GraphQL Explorer or go into the google/go-github reference docs and look for matching types such as "RepositoriesService".

The Code

Important! I use GitHub Copilot with VS Code. Without Copilot, it would have taken me 10 times longer to figure out exactly what to do. I use Copilot all the time but this case really made it worth the $10 a month. Not surprisingly, Copilot knows about the GraphQL types and really filled in a lot of the type information.

Each of the approaches: raw,go-github and shurcool perform the repo search and return the repo name,ID and stars count.

All access requires an authenticated client one way or another. The example code has the auth code and expects the environment variable GITHUB_TOKEN.

RAW

See pkg/raw/raw.go.

This approach cobbles up a POST request in GraphQL language, sends it to the API and gets a JSON result back. To unmarshall the JSON, a type must be set up that matches the return format. The code defines a type named repoData that mirrors the JSON structure and has the appropriate annotations.

Go-GitHub

See pkg/goog/goog.go

This approach is easiest. Just look up the desired type and methods in the go-github docs and code it up.

ShurcooL

See pkg/ghshurcool/ghshurcool.go.

This approach is in the middle between raw and go-github. It does require defining the query-search types. On the other hand, it allows fine grained access that the go-github API doesn't allow.

Note: The shurcooL/graphql package provides 'graphql' types including String, Float ID, Int and Int32. However, in the source comments and issues it says that the graphql types are not need. I had no problem using string, int, float directly.

Setup And Run

Auth Token

- You need a GitHub account.
- In order to access the GraphQL API, you need to get an auth token from your GitHub Account. The API requires authentication (The REST API does not). To get one, do the following:
    - login to your account
    - in the upper right corner, click on your profile picture
    - in the drop down list select "Settings"
    - on the Settings page, select "Developer Settings".
    - on the left menu, select "Personal Access Tokens"
    - select "Fine-grained tokens". (These allow more targeted permissions on the token)
    - generate a new token
        - by default, new tokens allow only read-only access to the API. That's enough for this exercise.
        - it may ask for your password or passkey to verify it's you
        - follow the instructions. Remember that once you generate the token, you need to copy it and store it somewhere (NOT IN A REPO). You won't be able to look at it again on the GitHub page.
        - For this exercise, the code will look for an environment variable named "GITHUB_TOKEN".

    - The token will look something like this "github_pat_blahblahblah..."

Install Go

The code here requires Go 1.18 or later (for go.work and module mode). If you are reading this, you probably already have Go, but if not, go to Go Download and Install.

Note : All the code here is built and tested on Linux Mint 21.2 Victoria. Compatible with Ubuntu 22.04LTA. But Go on any platform should work.

Clone The Repo

If you haven't already.

The repo with code

Run It

- printenv | grep GITHUB_TOKEN (just checking)
- $cd into the repo (top level)
- $go run . raw [owner name]
- $go run . goog [owner name]
- $go run . shurcool [owner name]

OR

- $cd into the repo (top level)
- $source test.sh (runs all three against the 'octocat' owner)