This week you have no actual SQL problem, per se, but instead are just focused on setting up the initial steps of your data generation project. I'll walk you through the individual steps below, but they closely mimic what we've done already either in homework on in class demonstrations. I've included a *lot* of text here, but it is all just hopefully supporting and walking you through the same setup I showed in class.

As per usual, I am creating a starting repository for you. If, for whatever reason, you can not add the organization repository for the `web2db` docker container, you could add the Dockerfile to this repository and deploy it from there. In order for this to work, I'll need to be approving your repositories as they are created (as you accept the assignment). If, when you go to add your repo, you don't see it, check with me! But ideally you'll be able to use the public repo in our class organization.

Assignment link: https://classroom.github.com/a/_mCBqgsP

1. To begin, you are going to need to come up with an idea of what you might want to scrape! I'm trying to leave this as open as possible, so that you can pursue something of joint interest to yourself and your project partner. The only real restriction is that you should plan to scrape data that is changing on at least a *mostly* daily basis. If it is something changing even more often, then you could get more data. But to get something interesting you want whatever you are scraping to at least change perhaps 5 days a week. Some ideas might include:

   - Weather data (OpenWeather has a free API)
   - Traffic data (TomTom has a freemium version that gives you *plenty* of requests per day)
   - News data (Perigon has a free version with up to 500 requests per month)
   - Currency rates
   - Sports club information
   - Event data (Ticketmaster has an API)
   - Stock information
   - Many games have an API that you can use to get information about player stats, etc.
   - The National Park Service has a free API
   - There are many APIs for tracking Covid-19 information
   - Apple's iTunes has a freely available API that can search its listings
   - There is a pretty comprehensive Space-X API on their rockets and spacecraft
   - Airline flight data

   The above are mostly APIs. Scraping websites directly may still work, but many websites that are frequently updated (and thus might be interesting to scrape) likely rely on a back-end server and probably use Javascript or similar to serve up the latest results. But if you find something interesting, pursue it!

   Make sure that the data that you and your project partner are individually scraping is data that, when combined, might be able to investigate an interesting question or explore an interesting relationship! In the template form in the repository, detail what website or API you will be scraping and describe any extra information you'll need to provide to the API (if applicable) to get the information you want. (Aka, what endpoint are you using? Do you need to provide extra parameters? Etc.)

2. Here you should test and ensure that you can scrape the data and write it to your *local* database. This will involve pulling in the `jrembold/web2db` Docker image and setting up your environment variables accordingly. You'll also need to create the necessary table in your local database, keeping in mind that the table needs a column named `raw_json`. Ensure that you can both run the Docker image as a one-shot scrape, AND that you can schedule a scrape to happen every other hour. In the template file for this problem, include your all your environmental variables *except* your `DATABASE_URL`. Upload a screenshot of the scraped data sitting in your SQL table.

3. Finally, you need to get this same system set up and running on Railway.app. Proceed to Railway.app and use your GitHub account to sign up. Initially this will grant you with $5 of credit and 500 free hours of use per month. You will always get $5 of free credit each month, even after you upgrade your account and enter a credit card number. 500 free hours will not get you through an entire month, but for what we are doing, $5 of free credit *should*. So the system should be effectively free to you, even if you'll want to upgrade your account with a credit card at some point this month so that the 500 hour limit doesn't stop your deployment.

   Following the tutorial we went through in class, create a new project, and add a PostgreSQL database to the project. Once this is done, ensure you can connect to it from your local client, and add whatever starting table you need to dump the raw JSON information into. Remember that adding a column with a `DEFAULT now()` parameter can be a nice way to track when new data is added without actually needing to populate it yourself! Then click in Railway to add a new component and select *GitHub Repo*. Assuming you signed in with your GitHub account, I've already approved our class organization, and hopefully you can see the organization `web2db` repo. If not, you'll have to add *this* repository, and I'll need to approve it before Railway will see it. Message me if this is what you are falling back on, though I'll try to be periodically checking. If all went well, you should see the repo for this assignment available for you to use. If you don't, let me know! But otherwise just select that repository, and, as soon as that goes through, you should see the new component pop up in Railway, which will then say "Deploying". It might take it a handful of minutes to build and deploy the image initially, so be patient! While it is deploying, ensure that you have set up all your necessary environment variables in the appropriate tab in Railway. Once your deployment it gets a green checkmark on it, check into your database to make sure the first element has been added to your table.

   For this problem, just uploading a screenshot of the scraped information within the table in Railway will suffice.

   Once you are sure information is being added to your table(s) in your remote database, you are done! *Make sure you check whenever the next database write would be*, to see that things are still being added!