Guide: Creating a GitHub Actions Workflow for Web Scraping

Let's consider an example web scraping task where we aim to collect the latest news headlines from a popular news website. We have written code using Python and the Beautiful Soup library to scrape the website's homepage and extract the headlines.

Suppose that this news site is updated frequently, and we want to obtain the headlines every hour. Then, we have the burden of needing to run this code every hour! But there is a better way. GitHub Actions allows us to schedule the scraping job to run at a specific time. By using GitHub Actions, we can automate this scraping task and receive the latest headlines without having to manually run the code so frequently.

Here are a few examples of scraping projects where utilizing GitHub Actions can be beneficial:

- News Aggregation: Automating the collection of news headlines or articles from various sources and saving them to a database or generating reports. GitHub Actions can schedule scraping tasks to run at specific times, ensuring you have the latest news data readily available for analysis or publication.
- Social Media Monitoring: Scraping social media platforms to gather data on trending topics, user sentiments, or specific keywords. With GitHub Actions, you can automate the scraping process to run at regular intervals, allowing you to monitor social media activity and extract valuable insights.
- Data Tracking and Price Monitoring: Tracking and scraping data from e-commerce websites, stock markets, or cryptocurrency exchanges. GitHub Actions can be configured to regularly fetch updated data, enabling you to monitor price changes, availability, or market trends automatically.

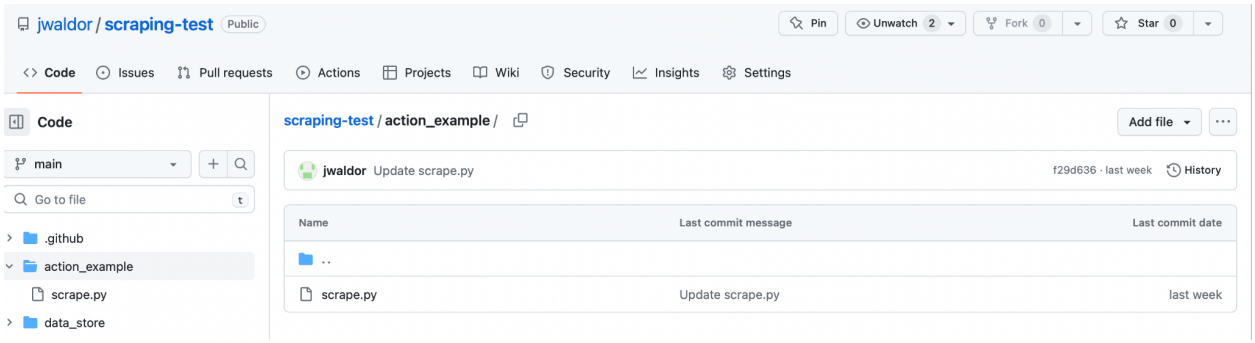
How to Use GitHub Actions

In this guide, we'll walk through the steps to create a GitHub Actions workflow that runs a web scraping script on an hourly basis and saves the scraped data to a file. The scraping algorithm scrapes from cnn.com. Let's get started:

Step 1: Create a new repository on GitHub.

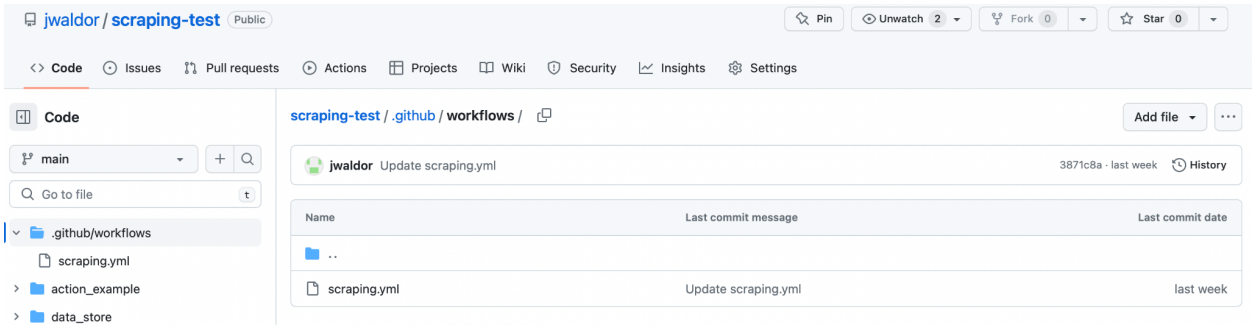
Step 2: Write the Web Scraping Code

1. Create a new file in your repository and name it scrape.py.
2. Copy and paste the following code into the scrape.py file: scrape.py code link
    a. This code pulls headlines from the CNN homepage. To learn the basics of web scraping, refer to the Scraping 101 notebook.



Step 3: Create the Workflow File

1. In your repository, create a new file named .github/workflows/scraping.yml. This will be your workflow file.
2. Copy and paste the following code into the scraping.yml file: [scraping.yml code link]



We now explain the key elements of the scraping.yml file above, such as the parts that determine the schedule of our task and provide the Python dependencies for our scraping script.

```
on: schedule: - cron: '0 * * * *'
```

- The `on` section defines the event that triggers the workflow. In this case, it is set to schedule, indicating that the workflow will run on a specific schedule.
- The `cron` field within `schedule` uses the cron syntax, which is a way of specifying the schedule. Here, `'0 * * * *'` represents running the workflow at the top of every hour (i.e., 0 minutes past the hour). It follows a specific format that consists of five fields, each representing different aspects of the schedule.

Here are a few examples to help you understand the cron syntax. For more information, you can go here: https://crontab.guru/.

- `* * * * *`: Runs the task every minute.

- `0 * * * *`: Runs the task at the start of every hour.
- `0 12 * * *`: Runs the task once daily at 12:00 PM.
- `0 0 * * 1`: Runs the task every Monday at midnight.
- `0 0 1 1 *`: Runs the task once a year on January 1st at midnight.

```
- name: Install dependencies run: | pip install beautifulsoup4 pip install
requests
```

- This step runs shell commands using the `run` keyword to install the required Python dependencies for scrape.py.
- It uses `pip install` to install the `beautifulsoup4` and `requests` packages.

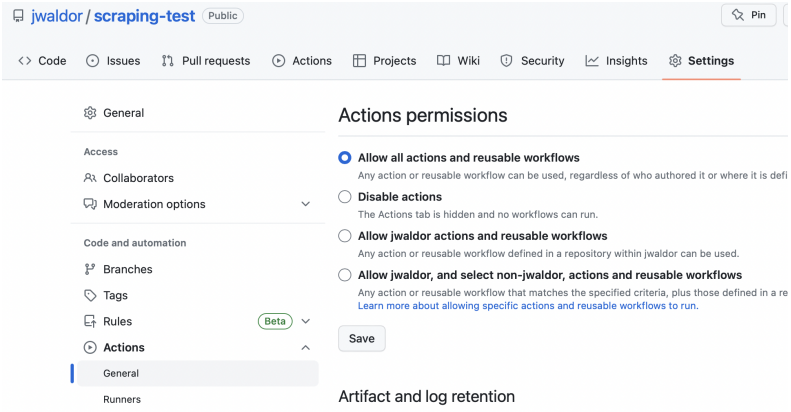
```
- name: Run scraping code and save to file run: python action_example/scrape.py
--output action_example/data.txt
```

- The final step executes the scraping code by running the `scrape.py` script.
- The `--output` argument is used to specify the output file path.

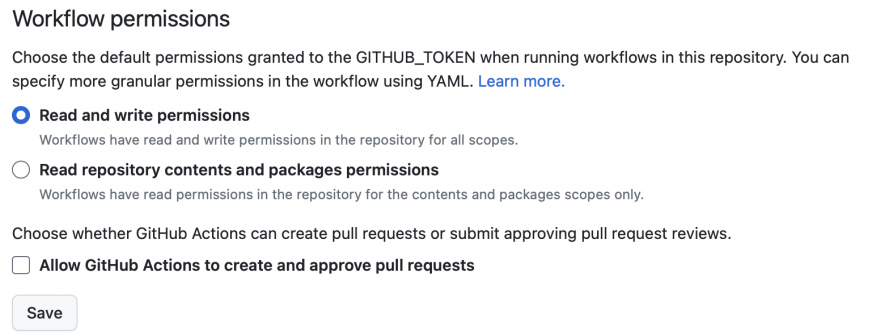
Each step within the steps section performs a specific action, such as setting up the environment, installing dependencies, and running the scraping code. By defining these steps, the workflow can automate the entire process of scraping the data from the specified URL and saving it to a file.

Step 4: Change repository settings to give workflows the ability to push files to the repository.

1. Go to the Settings section in the repository.
2. On the left hand side, click on Actions -> General.



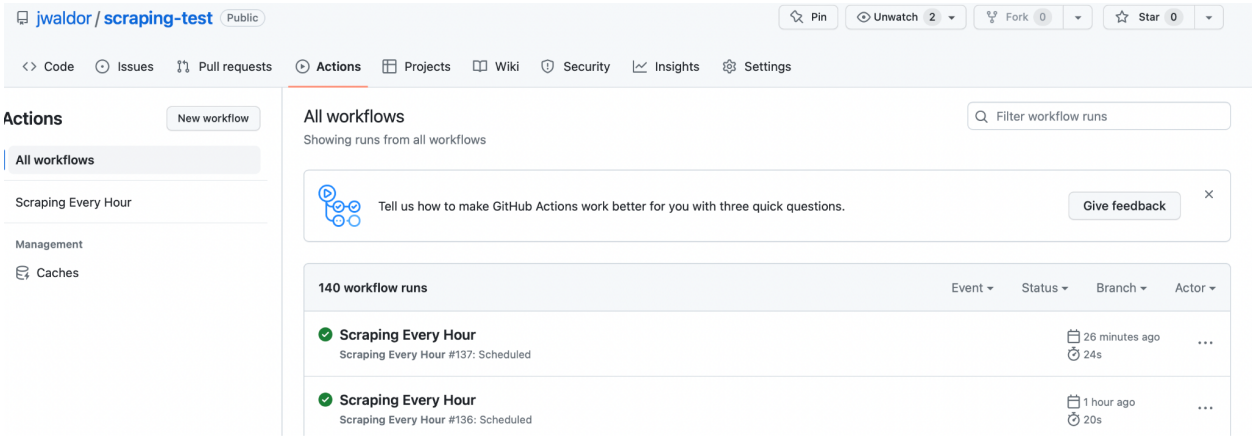
3. Scroll down to Workflow Permissions. Select Read and Write Permissions, and then click Save.



Step 5: Commit and Push the Workflow

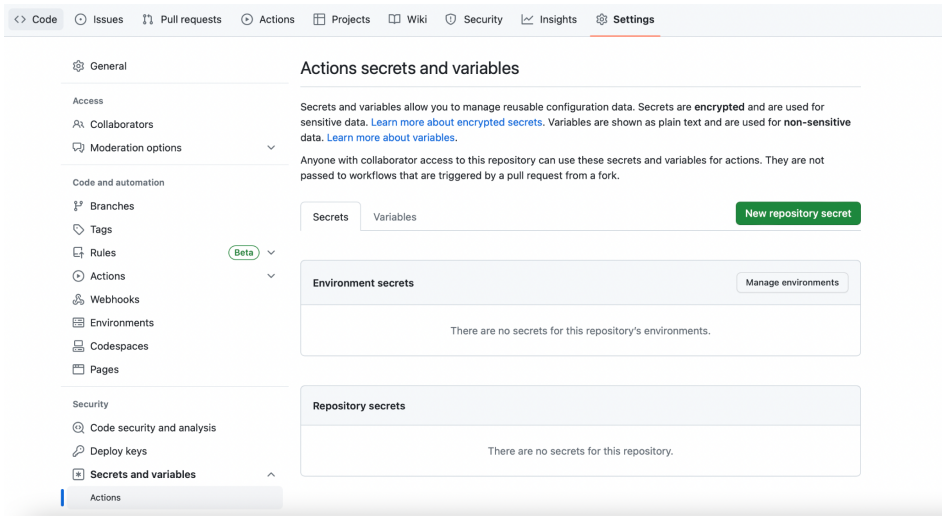
1. Save the changes to the `scraping.yml` file.
2. Commit the changes to your repository.
3. Push the commit to trigger the workflow.

Now, you can check if your workflow is running properly by clicking the Actions tab on your repository:



A Note on Secrets

Sometimes you want to use an API key in your scraping code. This is considered using a “Secret” in your code, since this is information that people generally want to protect. To create a secret, you can go to the Actions->Secrets and Variables and click “New repository secret.”



Then, you'll be able to use the scraping.yml file to give your scraping code access to the secret by adding the following step to scraping.yml:

```
    - name: Set up environment variables
        env:
          API_KEY: ${{ secrets.API_KEY }}
```

Here, we allow the Python code to access the secret API_KEY. To load the secret into our Python script, we use the following lines of code:

```
import os
api_key = os.getenv('API_KEY')
```

That's it! You now know how to set up Actions in GitHub for web scraping. You can customize the scraping frequency in scraping.yml, and you can modify scrape.py to carry out any scraping task of your choosing. You can check out the full example repository here. For more information, you can also read the official documentation for GitHub Actions. Thanks!