COMP7940 Cloud Computing

2022/23 S2 Lab 6 Git action

Instructor	Dr. Qichen Wang	qcwang@hkbu.edu.hk
Teaching Assistant	Mr. Zhengheng Tang	zhtang@comp.hkbu.edu.hk
Teaching Assistant	Mr. Zhen Ye	cszhenye@comp.hkbu.edu.hk

Objective:

Throughout this lab you will be able to:

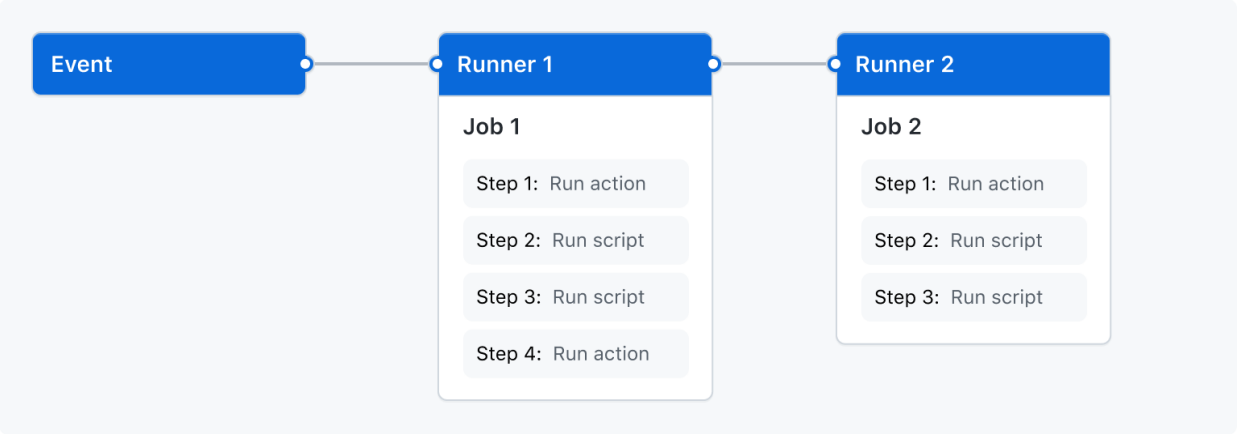
- Create a GitHub Action and use it in a workflow

Introduction:

GitHub Actions helps you to automate tasks within your software development life cycle. GitHub Actions is event-driven, meaning that you can run a series of commands after a specified event has occurred. For example, every time someone creates a pull request for a repository, you can automatically run a command that executes a software testing script.

The components of GitHub Actions

Below is a list of the multiple GitHub Actions components that work together to run jobs. You can see how these components interact with each other.



Workflows

A workflow is an automated procedure that you add to your repository. Workflows are made up of one or more jobs and can be scheduled or triggered by an event. The workflow can be used to build, test, package, release, or deploy a project on GitHub.

Events

An event is a specific activity that triggers a workflow. For example, activity can originate from GitHub when someone pushes a commit to a repository or when an issue or pull request is created. You can see list of events that can be used to trigger workflows here.

Jobs

A job is a set of steps that execute on the same runner. By default, a workflow with multiple jobs will run those jobs in parallel. You can also configure a workflow to run jobs sequentially.

Steps

A step is an individual task that can run commands in a job. A step can be either an *action* or a shell command. Each step in a job executes on the same runner, allowing the actions in that job to share data with each other.

Actions

Actions are standalone commands that are combined into *steps* to create a *job*. Actions are the smallest portable building block of a workflow. You can create your own actions, or use actions created by the GitHub community. To use an action in a workflow, you must include it as a step.

Runners

A runner is a server that has the GitHub Actions runner application installed. You can use a runner hosted by GitHub, or you can host your own. A runner listens for available jobs, runs one job at a time, and reports the progress, logs, and results back to GitHub.

Create an example workflow

GitHub Actions uses YAML syntax to define the events, jobs, and steps. These YAML files are stored in your code repository, in a directory called `.github/workflows`.

You can create an example workflow in your repository that automatically triggers a series of commands.

1. In your repository, create the `mkdir .github` and `mkdir .github/workflows/` directory to store your workflow files.
2. Under the directory `.github/workflows/`, create a new file called `git-actions.yml` that contains the following code:

```
   name: my-actions                # the name of this work flow
   on: [push]
   # Specify the event that automatically triggers the workflow file. This example
uses the push event, so that the jobs run every time someone pushes a change to the
repository

   jobs:                            # gather all the jobs that run in the
workflow
     my-helloworld-job:
       runs-on: ubuntu-latest          # Set the type of machine to run on
       steps:
         - uses: actions/checkout@v2   # Step 1: Checks out a copy of your
repository on the ubuntu-latest machine

         - name: Say Hello World       # Step 2: Type the word 'Hello World'
           run: echo 'Hello World'

         - name: View a file           # Step 3: Two sub-steps involved
           run: |
              echo 'Viewing the file: '
              cat requirements.txt
```

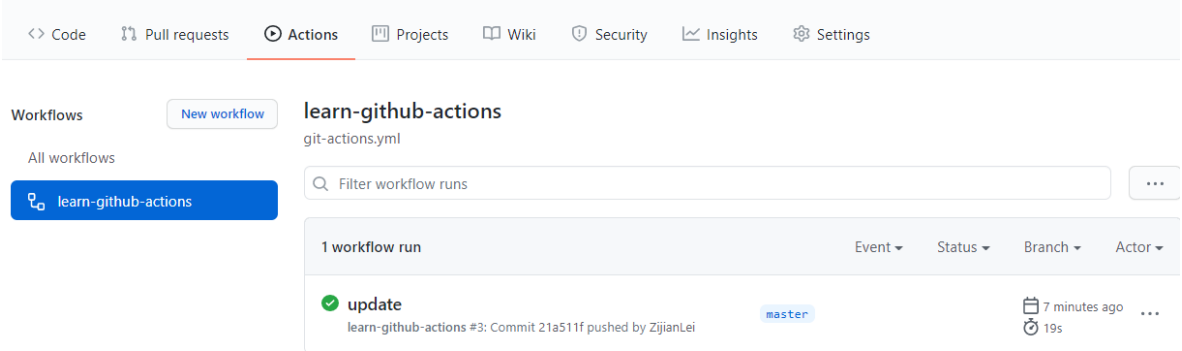
3. Commit these changes and push them to your GitHub repository.

Your new GitHub Actions workflow file is now installed in your repository and will run automatically each time someone pushes a change to the repository.

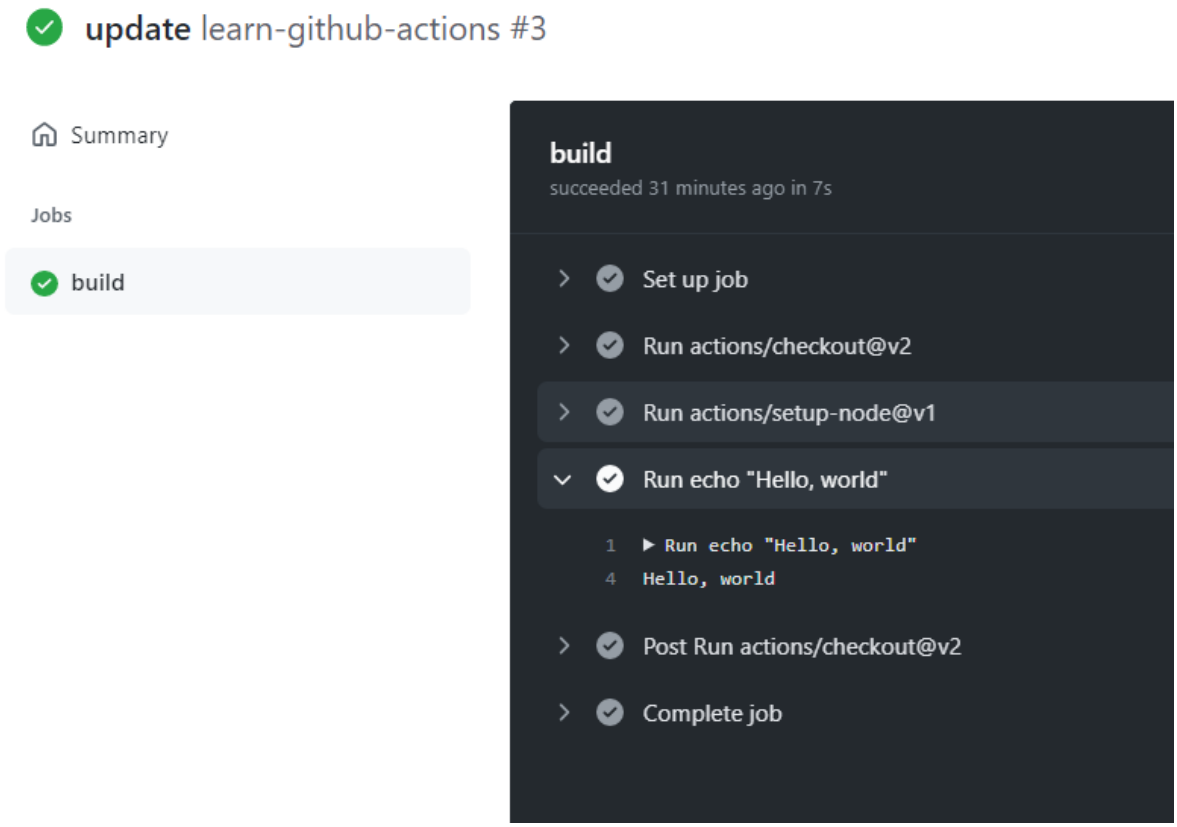
Viewing the job's activity

Once your job has started running, you can see a visualization graph of the run's progress and view each step's activity on GitHub.

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Actions**.
3. In the left sidebar, click the workflow you want to see.



4. Under "Workflow runs", click the name of the run you want to see.
5. Under **Jobs** or in the visualization graph, click the job you want to see.
6. View the results of each step.



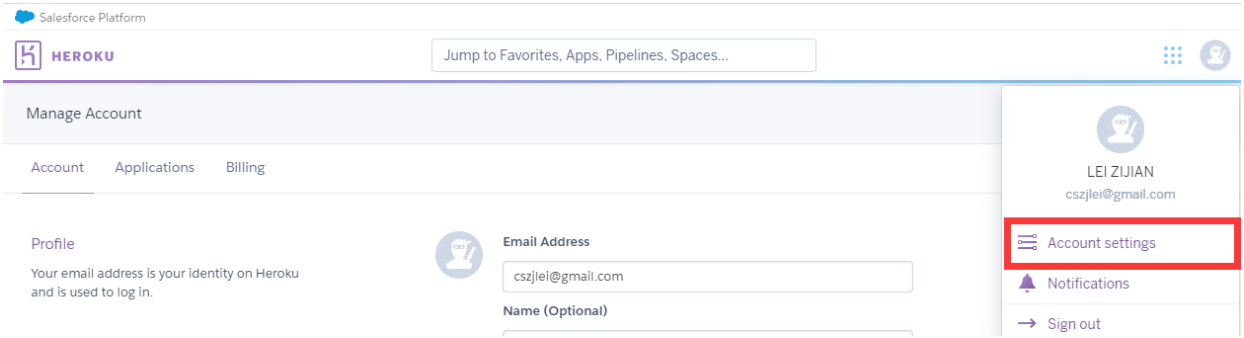
Automatically Deploy to Heroku

Now we add the following `deploy.yml` to the git repository. The source code will be pushed to Heroku when you push the code from your local machine to GitHub.

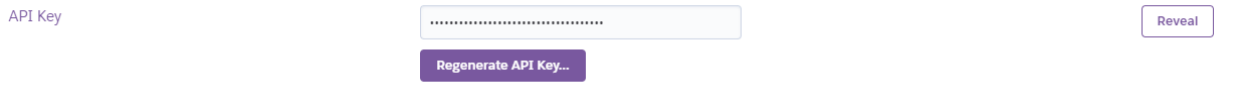
```yaml
name: Deploy
on:
  push:
    branches:
      - main
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
        with:
          fetch-depth: 0
      - name: Deploy
        env:
          HEROKU_API_KEY: ${{secrets.HEROKU_API_KEY}}
```

```
          HEROKU_APP_NAME: "comp7940chatbot"  # Rename this to the app of your
Heroku app.
        run: git push -f
https://heroku:$HEROKU_API_KEY@git.heroku.com/$HEROKU_APP_NAME.git main
```

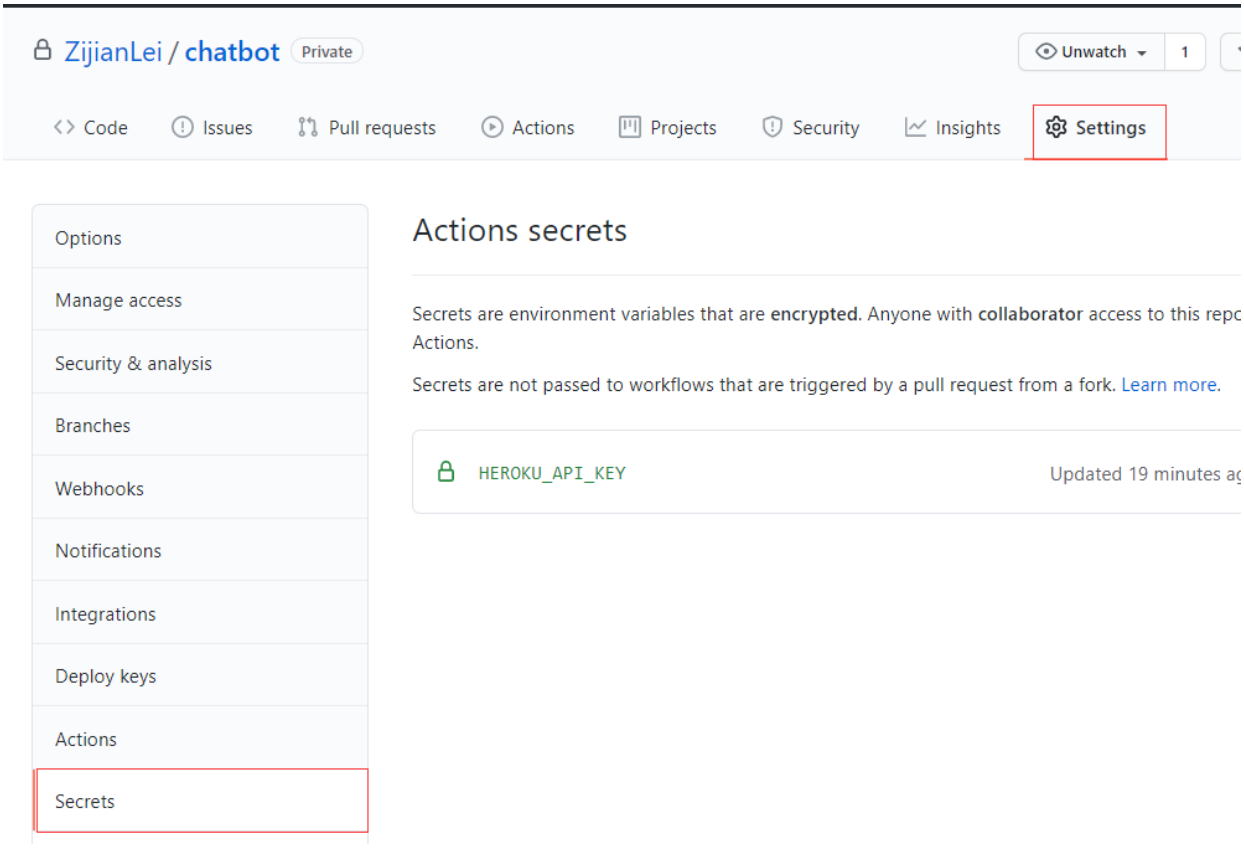
Now you need to set the `HEROKU_API_KEY` for this Git Actions. First, go to your Heroku account and go to `Account Settings`.



Scroll to the bottom until you see API Key. Copy this key and go to your project's repository on GitHub.



Then in your GitHub Repo, go to `Settings -> Secrets` and click on "New Secret". Then enter `HEROKU_API_KEY` as the name and paste the copied API Key as the value.



Reference:

Please read more to learn how GitHub Actions works, if interested: https://docs.github.com/en/actions/learn-github-actions/introduction-to-github-actions

---

Write up:

Answer the following questions and submit them to moodle:

1. Identify which ONE command does the job to upload/deploy your chatbot to Heroku.
2. How does your GitHub repo related/connected to your Heroku app and your local repo? Please draw them into a single diagram and explain it briefly.
3. Explain why it is a bad idea to use config.ini to store our password and how the current setting different than before?
4. Prove that you have finished lab 6. Make some screen captures of your GitAction and explain them.
5. GitHub Actions is far more useful than just printing Hello World or pushing code to Heroku, the following package: `aufdenpunkt/python-safety-check` allows you to check the security of your python source code. Please revise your YAML file to add this step into your workflow. Show us the updated YAML.