Github Action Exercices

Events

(Event List)

For this part, you will have to trigger the workflow under specific conditions. Each item on the list correspond to a particular scenario where the workflow should (*or should not*) run.

Push

1. *Create and publish 3 branches, "dev", "staging", "feat/feature-1"*
2. When you have a push on the branch "dev" only
3. When you have a push on any branch **except** "staging"
4. When you have a push on any branch where the name starts with "feat"
5. When only the files with a ts extension have been modified
6. When a specific folder have been modified

Pull request

1. When someone **creates** a PR
2. When someone **closed** a PR

Scheduled

Help

1. Every 5 minutes
2. Every 1h 30m
3. Every Tuesday at 3am

Push & PR

1. *Create two jobs inside a new workflow*
2. When you have a push, execute both job
3. When you have a PR, execute only the first job.
4. *Create a **dev** branch, and inside it create a dummy change.txt file. Put your name in it. Publish this branch*
5. Create a pull request from the **dev** branch to **main**. You should see the first job executing directly inside the PR (from the github UI)

Manual & Custom

1. When you click on *run workflow* button in the UI
2. *Create two workflows*
3. Trigger the second workflow from the first one, and pass the github event name from the first one to the second with a variable named "firstJobEvent"

Troubleshooting tools & logs

Install the nektos/act tool to run your workflows locally.

1. *Once **act** is installed, run the command from your cli (`act`) and let it download the docker image it needs.*
2. *Create a new repository secret called **ACTIONS_STEP_DEBUG** and set its value to true*
3. Re-run your last workflow from the github action UI
4. Get a particular line from the job's output and extract a link that points to that line. Save it in a result.
5. Download the logs for an entire job
6. *Create a new repository secret called **ACTIONS_RUNNER_DEBUG** and set its value to true*
7. *Remove the **ACTIONS_STEP_DEBUG** secret*
8. Re-run the last workflow
9. Download the logs from the latest job, and compare the two debug mode. What's the major difference ?
10. Run your workflow with act. Don't forget to specify the event type.

Runners

Here, you will have to compose with the runner to achieve the demanded results. A node app has been added, in the `node-app` directory

Job's default

1. Write a workflow that installs dependancies and runs the test. Specify a default folder for all the steps inside the job.

Matrix

1. Create a matrix strategy for a node js app that will install dependancies for node 12, 14, 16
2. Create a second matrix strategy to test the node app on ubuntu and windows

Docker image runner

1. Now, instead of installing node js with the action, use a node image for the tests of the node-app

Self-hosted runners

1. Download the self-hosted runner script
2. Create a workflow that will install the node dependancies, test and run the app (`npm start`)
3. Look at the result from the job. Why is this a bad idea for public repositories ?

Environnements

1. In your repo's settings, create two environnements. One for production, one for staging.
2. Write a workflow that deploy the joke app to heroku on a push to the main branch
3. Write a workflow that deploy the joke app to heroku on a push to the dev branch (on a different heroku app)
4. For each workflow, specify the corresponding environnement and the url of the app you created
5. For the production environnement, set the following secret : `JOKE_URL=https://v2.jokeapi.dev/joke/Programming`
6. For the staging environnement, set the following secret : `JOKE_URL=https://v2.jokeapi.dev/joke/Any?blacklistFlags=racist,sexist`
7. Use the secret to incorporate different joke url depending on the environnement (Help)