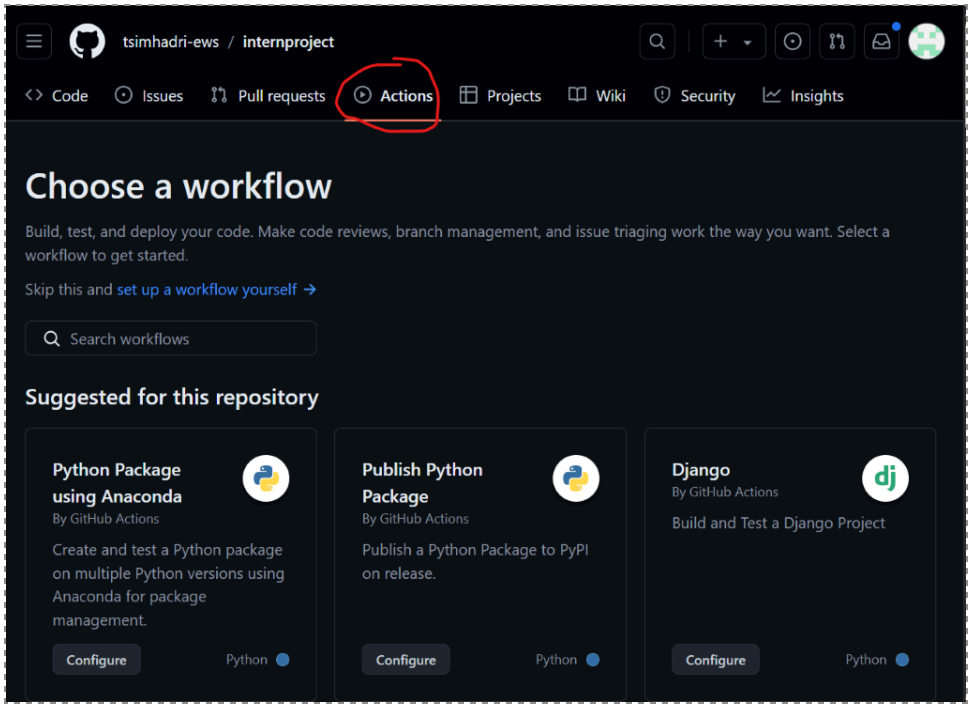
MLOps L2 - Setting up GitActions

Go to a repo on Github and select Actions



1. Select “Skip this and set up a workflow yourself”
2. Create a main.yaml file for the GitAction
3. If you need passwordless access to Kubeflow, follow the adding a secret to GitHub Documentation
4. Copy/create a file structured similar on that on the next page:

```
name: CI Name of the workflow that is create change to
CI_[NameOfPipeline]

# Controls when the workflow will run
on:

  push:
     branches: [ "main" ] Branch that changes are pushed to that
will run
     paths: Files that when changed with run the Github Action
     - 'Pipeline/Malware/Production/read_file.py'
     - 'Pipeline/Malware/Production/train_op.py'


  # Allows you to run this workflow manually from the Actions tab
  workflow_dispatch: Do not need to include, I added for debugging

# A workflow run is made up of one or more jobs that can run
sequentially or in parallel
jobs:
  # This workflow contains a single job called "build" This is the
version of Ubuntu we are using
  build:
     # The type of runner that the job will run on
     runs-on: ubuntu-latest

     # Steps represent a sequence of tasks that will be executed as
part of the job
     steps:
     # Checks-out your repository under $GITHUB_WORKSPACE, so your
job can access it
     - uses: actions/checkout@v4

     - name: Set up Python Python version getting used
     uses: actions/setup-python@v4
     with:
          python-version: '3.7'

Dependencies are installed for each script, then the script is ran
     - name: Install dependencies
     working-directory: ./Pipeline/Malware/Production
     run: pip install -r main_requirements.txt
 Python files that are run as part of the pipeline
```

```
    - name: Check condition
    working-directory: ./Pipeline/Malware/Production
    run: python check_condition.py

    - name: Run read
    working-directory: ./Pipeline/Malware/Production
    run: python read_file.py


    - name: Run train
    working-directory: ./Pipeline/Malware/Production
    run: python train_op.py

    - name: Install dependencies
    working-directory: ./Pipeline/Malware/Production
    run: pip install -r compile_requirements.txt
```

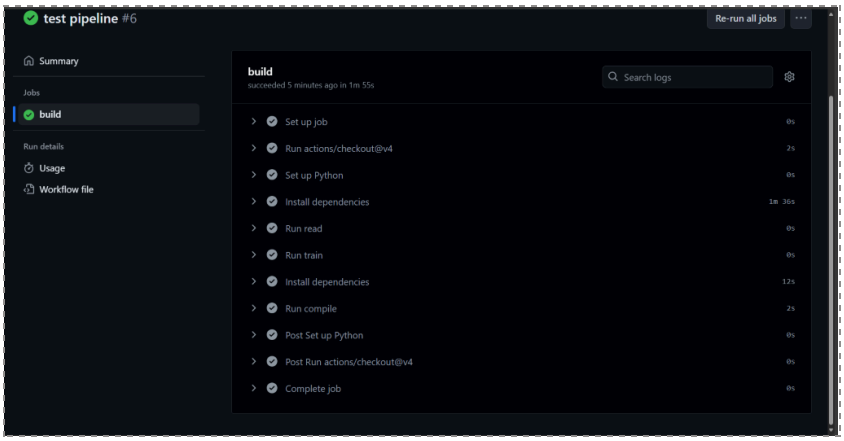
Kubeflow credentials are stored in github secrets

```
    - name: Run compile
    working-directory: ./Pipeline/Malware/Production
    env:
         KUBEFLOW_USERNAME: ${{ secrets.USER }}
         KUBEFLOW_PASSWORD: ${{ secrets.PASSWORD }}
         KUBEFLOW_TOKEN: ${{ secrets.TOKEN }}
    run: python compile.py
```

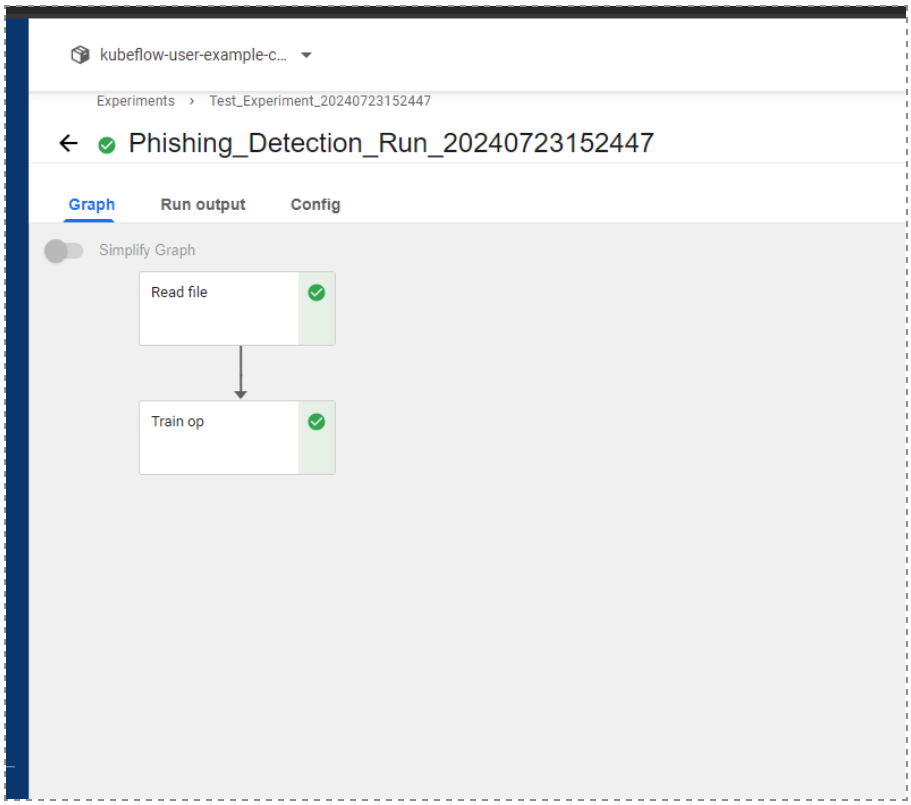
The workflow you just created will be saved in .github/workflows on the specified branch.

When changes are made or read or train [or the file specified in the main.yaml], the pipeline will run

This is what the run looks like in GitActions:



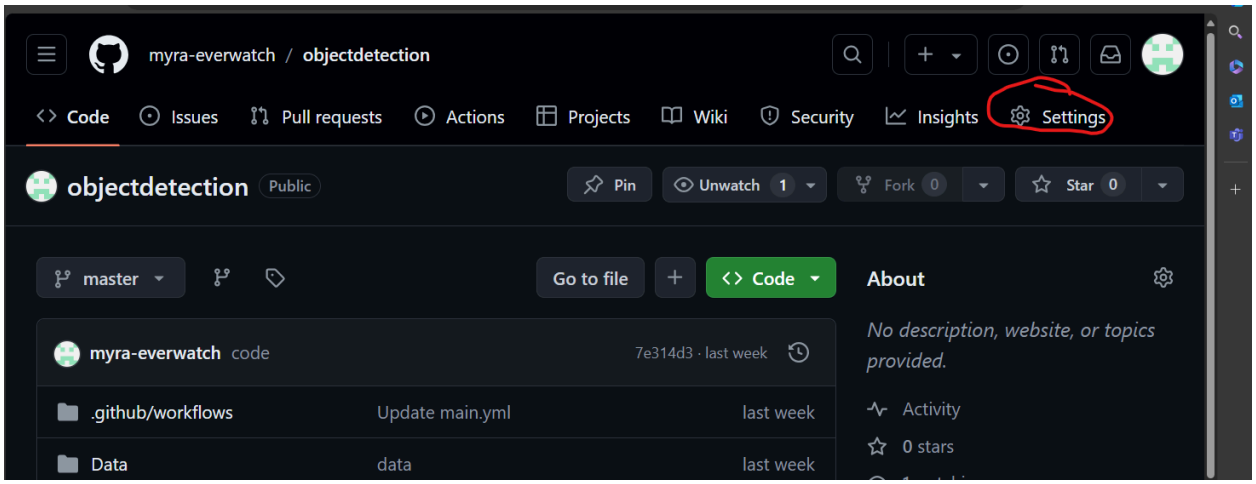
After the run builds the pipeline, it will run the pipeline in Kubeflow:



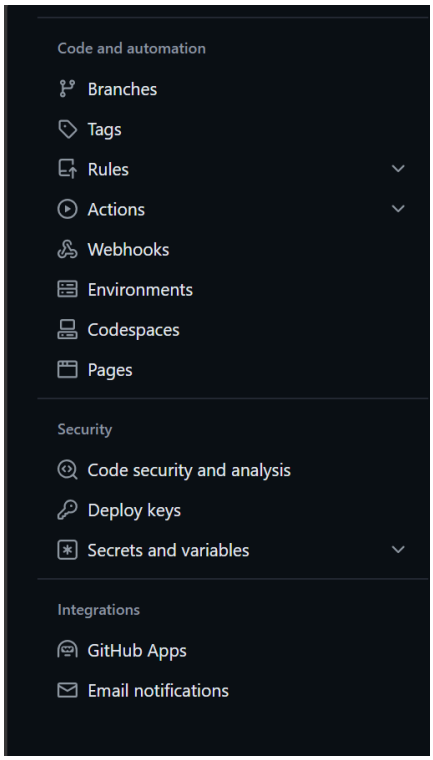
MLOps L2 - Storing a secret in GitHub

Prerequisite: The user must be the owner of the repo to set it up.

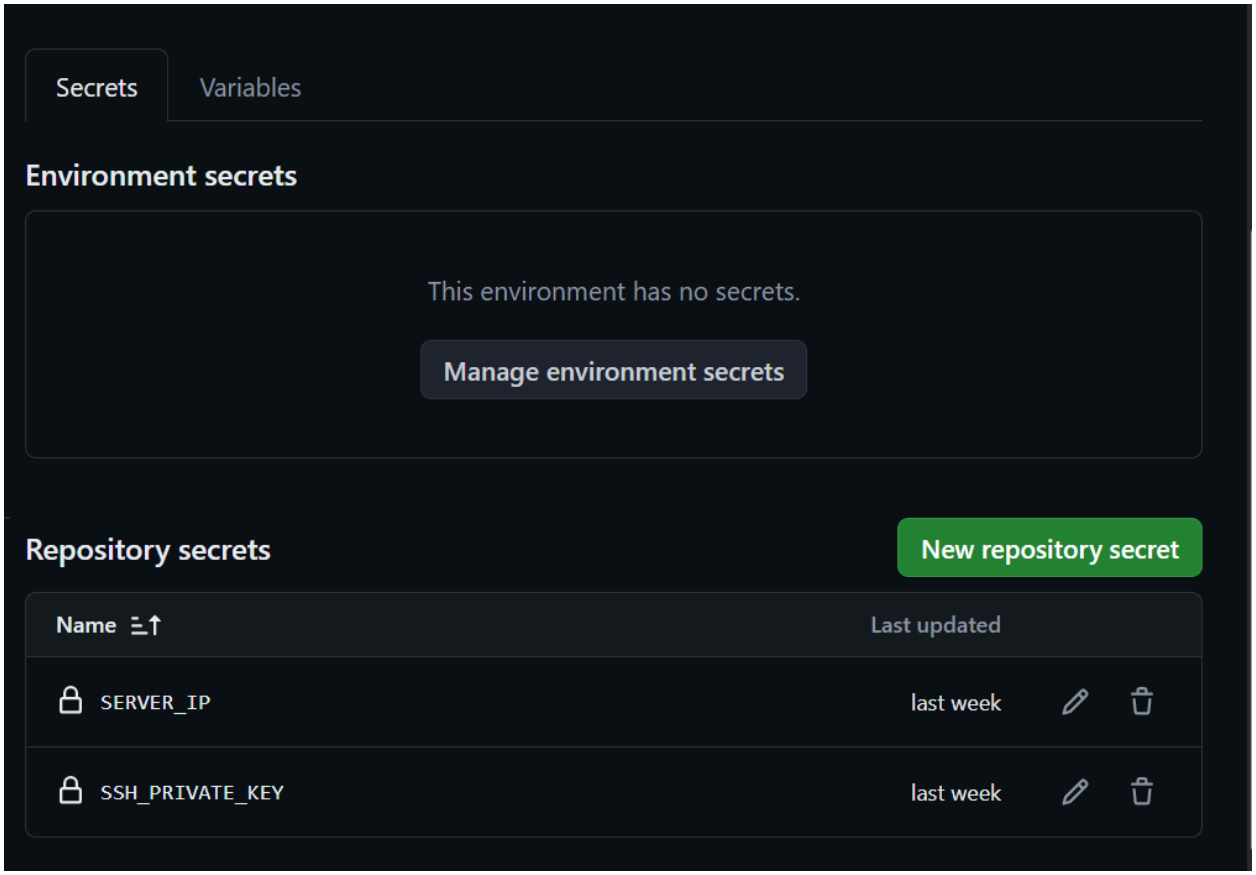
Go to a repo on Github and select “Settings”



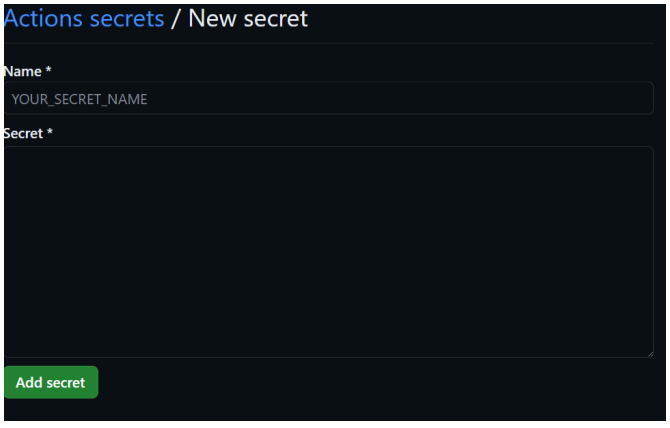
Then select “Secrets and variables” from the left side menu



1. Click Actions underneath Secrets and variables.
2. Select New repository secret



Add the name of your secret and the secret.



Now you can programmatically read in the secret by referencing the name you set.

For example, this code in a GitAction would retrieve the secrets values of secrets named, “USER”, “PASSWORD”, and “TOKEN”
Secrets are referenced using the “secrets.” syntax

```
KUBEFLOW_USERNAME: ${{ secrets.USER }}
KUBEFLOW_PASSWORD: ${{ secrets.PASSWORD }}
KUBEFLOW_TOKEN: ${{ secrets.TOKEN }}
```