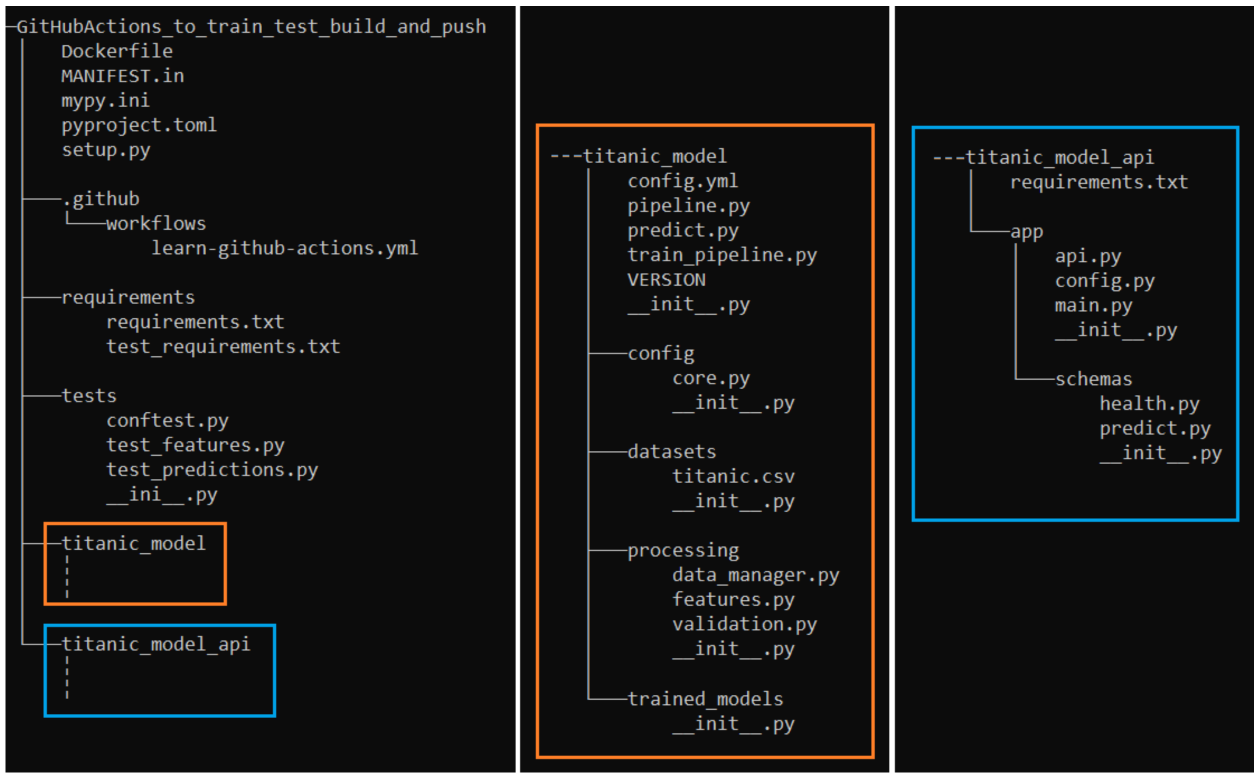
GitHub Actions to push Docker Image

Setup

Ensure that you have a GitHub account, and an active DockerHub account.

Steps to automate train, test, build, and push docker image

1. Download the folder 'GitHubActions_to_train_test_build_and_push' shared along with this document. It contains all the files related to training, testing, package building, and application dockerizing for Titanic dataset.
2. Directory structure is similar to below.



```
─GitHubActions_to_train_test_build_and_push
 │   Dockerfile
 │   MANIFEST.in
 │   mypy.ini
 │   pyproject.toml
 │   setup.py
 │
 ├───.github
 │   └───workflows
 │           learn-github-actions.yml
 │
 ├───requirements
 │       requirements.txt
 │       test_requirements.txt
 │
 ├───tests
 │       conftest.py
 │       test_features.py
 │       test_predictions.py
 │       __ini__.py
 │
 ├───titanic_model
 │   ¦
 │   ¦
 │
 └───titanic_model_api
     ¦
     ¦
```

```
---titanic_model
   │   config.yml
   │   pipeline.py
   │   predict.py
   │   train_pipeline.py
   │   VERSION
   │   __init__.py
   │
   ├───config
   │       core.py
   │       __init__.py
   │
   ├───datasets
   │       titanic.csv
   │       __init__.py
   │
   ├───processing
   │       data_manager.py
   │       features.py
   │       validation.py
   │       __init__.py
   │
   └───trained_models
           __init__.py
```

```
---titanic_model_api
   │   requirements.txt
   │
   └───app
       │   api.py
       │   config.py
       │   main.py
       │   __init__.py
       │
       └───schemas
               health.py
               predict.py
               __init__.py
```

3. Note that a yml file is present at .github/workflows/ path that contains the details of jobs to execute.

   Content of yml file:

```yaml
name: Push a Docker Image
on:
  push:
    branches:
      - main
  workflow_dispatch:

jobs:
  train:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - name: Set up Python 3.10
      uses: actions/setup-python@v3
      with:
        python-version: '3.10'
    - name: Install dependencies
      run: pip install -r requirements/requirements.txt
    - name: Train and save pipeline
      run: python titanic_model/train_pipeline.py
    - uses: actions/upload-artifact@v2
      with:
        name: my-trained-pipeline
        path: titanic_model/trained_models/*.pkl

  test:
    needs: train
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - name: Set up Python 3.10
      uses: actions/setup-python@v3
      with:
        python-version: '3.10'
    - uses: actions/download-artifact@v2
      with:
        name: my-trained-pipeline
    - run: mv *.pkl titanic_model/trained_models/
    - name: Install dependencies
      run: pip install -r requirements/test_requirements.txt
    - name: Test with pytest
```

```yaml
      run: pytest

  build:
    needs: [train, test]
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - name: Set up Python 3.10
      uses: actions/setup-python@v3
      with:
        python-version: '3.10'
    - uses: actions/download-artifact@v2
      with:
        name: my-trained-pipeline
    - run: mv *.pkl titanic_model/trained_models/
    - name: Install dependencies
      run: pip install --upgrade build
    - name: Build package
      run: python -m build
    - uses: actions/upload-artifact@v2
      with:
        name: my-build-package
        path: dist/*.whl

  push-image:
    needs: [train, test, build]
    runs-on: ubuntu-latest
    steps:
    - name: Repo Checkout
      uses: actions/checkout@v2

    - uses: actions/download-artifact@v2
      with:
        name: my-build-package
    - run: mv *.whl titanic_model_api/

    - name: Login to Docker hub
      env:
        DOCKER_USER: ${{ secrets.DOCKER_USER_NAME }}
        DOCKER_PASSWORD: ${{ secrets.DOCKER_PASS_TOKEN }}
      run: docker login -u $DOCKER_USER -p $DOCKER_PASSWORD
  - name: Build the Docker image for Fastapi app
    env:
      DOCKER_USER: ${{ secrets.DOCKER_USER_NAME }}
    run: docker build . -f Dockerfile -t $DOCKER_USER/titanic-fastapi:latest

  - name: Push the Docker Image
    env:
      DOCKER_USER: ${{ secrets.DOCKER_USER_NAME }}
    run: docker push $DOCKER_USER/titanic-fastapi

  - name: Logout to Docker hub
    run: docker logout
```

In the above workflow, 4 jobs are defined:

- **train:** to train and save the pipeline.
  Once trained, the saved pipeline (pickle file) is uploaded as an artifact, using upload artifact action (`uses: actions/upload-artifact@v2`), so that it can be shared with other jobs.
- **test:** to run test cases.
  To make sure this job runs only after the train job is completed, the needs key pair is used (`needs: train`). Also, the saved pipeline artifact is downloaded using download artifact action (`uses: actions/download-artifact@v2`). Once downloaded, the pickle file has to be present inside the `trained_models` directory. Move it by running the mv command (`run: mv *.pkl titanic_model/trained_models/`).
- **build:** to build the distributable package.
  To build the package, python's build module is used (`run: python -m build`). Once the package (wheel file) is created, it is uploaded as an artifact.
- **push-image:**
  Within this job, first the wheel file artifact is downloaded and moved to the `titanic_model_api` directory. Then we login to DockerHub with docker login command and our docker credentials (`run: docker login -u $DOCKER_USER -p $DOCKER_PASSWORD`). Note that the docker credentials are stored in GitHub secrets, and are read as environment variables using env key pair (

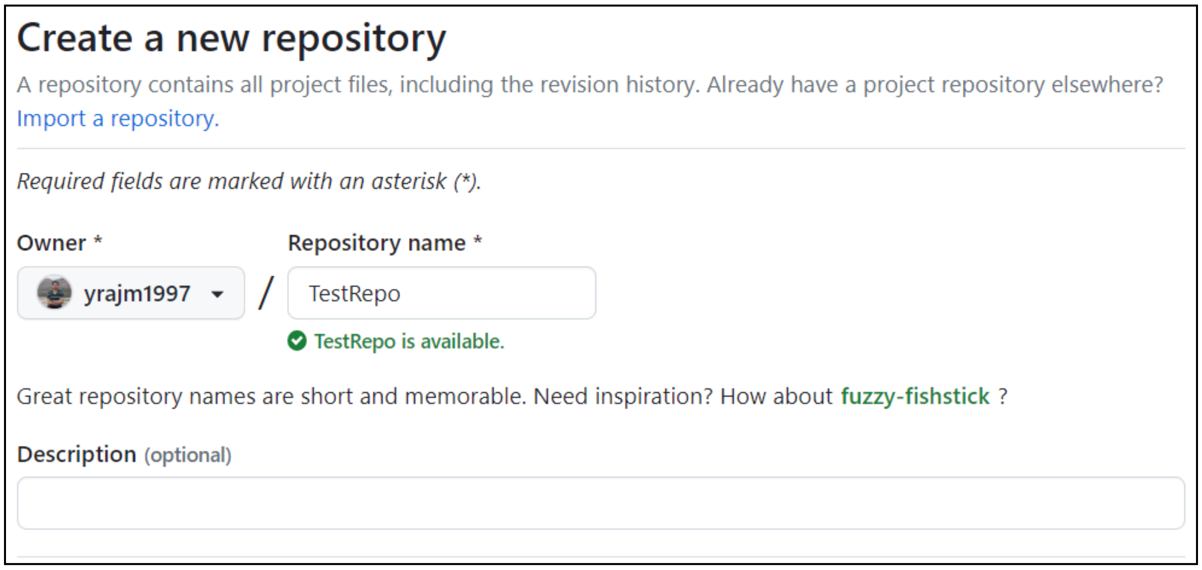
```
env:
  DOCKER_USER: ${{ secrets.DOCKER_USER_NAME }}
  DOCKER_PASSWORD: ${{ secrets.DOCKER_PASS_TOKEN }}
```
).

Later, the image was built for the Fastapi application as per the Dockerfile (`run: docker build . -f Dockerfile -t $DOCKER_USER/titanic-fastapi:latest`), and pushed to DockerHub (`run: docker push $DOCKER_USER/titanic-fastapi`).

4. The Dockerfile used is shown below.

```
# pull python base image
FROM python:3.10
# copy application files
ADD /titanic_model_api /titanic_model_api/
# specify working directory
WORKDIR /titanic_model_api
# update pip
RUN pip install --upgrade pip
# install dependencies
RUN pip install -r requirements.txt
# expose port for application
EXPOSE 8001
# start fastapi application
CMD ["python", "app/main.py"]
```

5. To build a complete pipeline, first create a new GitHub repository.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

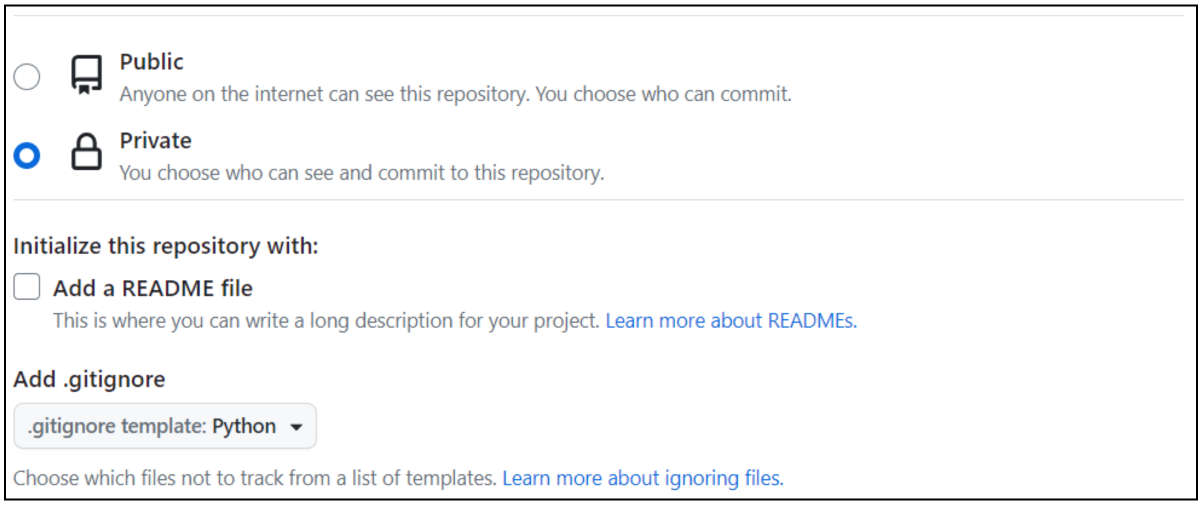
Required fields are marked with an asterisk (*).

Owner * yrajm1997 / Repository name * TestRepo

TestRepo is available.

Great repository names are short and memorable. Need inspiration? How about fuzzy-fishstick ?

Description (optional)



Public
Anyone on the internet can see this repository. You choose who can commit.

[✓ Private]
You choose who can see and commit to this repository.

Initialize this repository with:

Add a README file
This is where you can write a long description for your project. Learn more about READMEs.

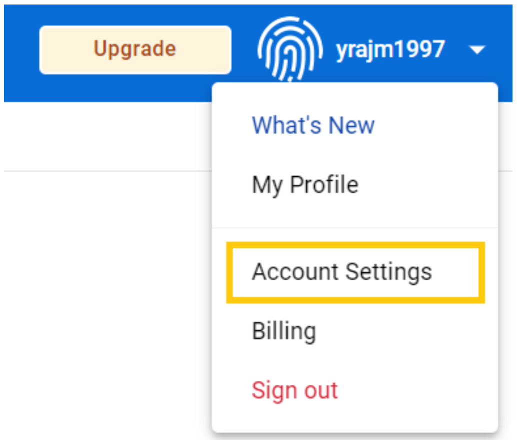
Add .gitignore

.gitignore template: Python

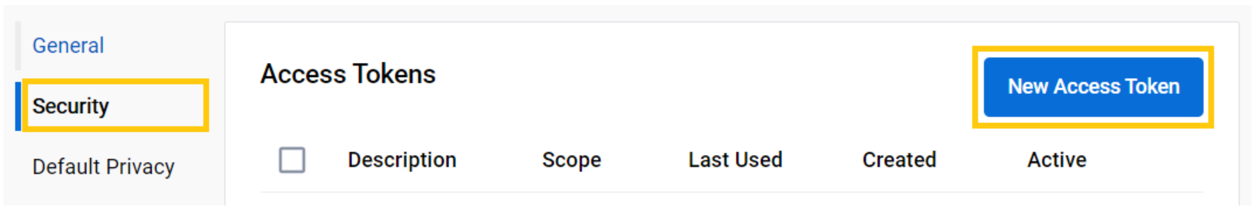
Choose which files not to track from a list of templates. Learn more about ignoring files.

6. Generate Access Token for the DockerHub Account.

   Go to your DockerHub Account > *Account Settings*.

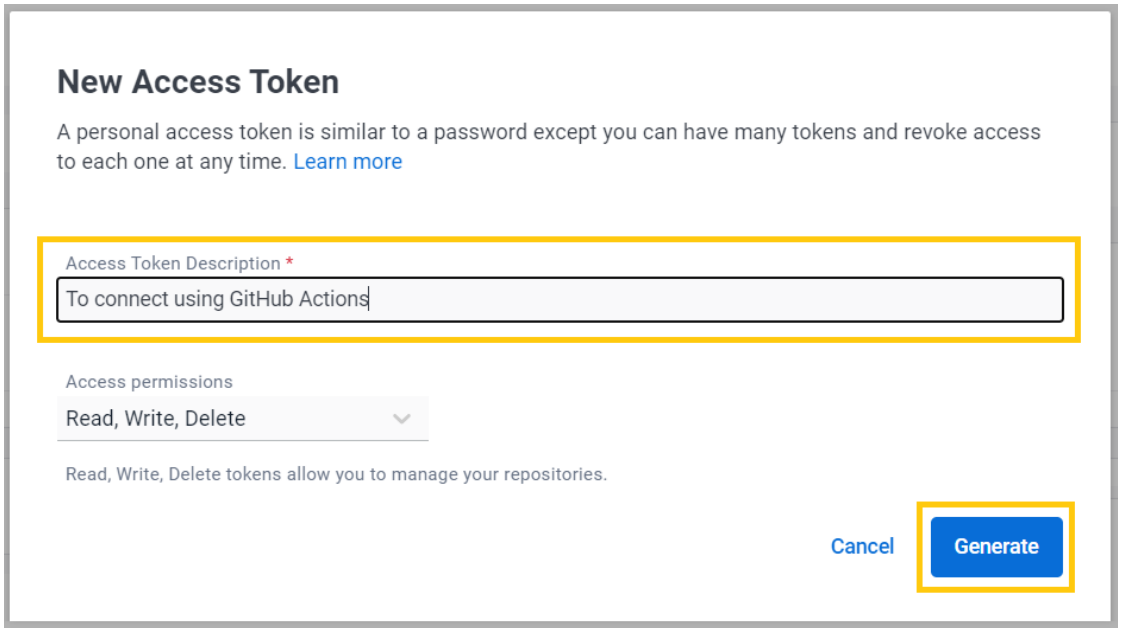


   Select *Security* > *New Access Token*.



	Description	Scope	Last Used	Created	Active

   Give description, and click *Generate*.


New Access Token

A personal access token is similar to a password except you can have many tokens and revoke access to each one at any time. Learn more

Access Token Description *

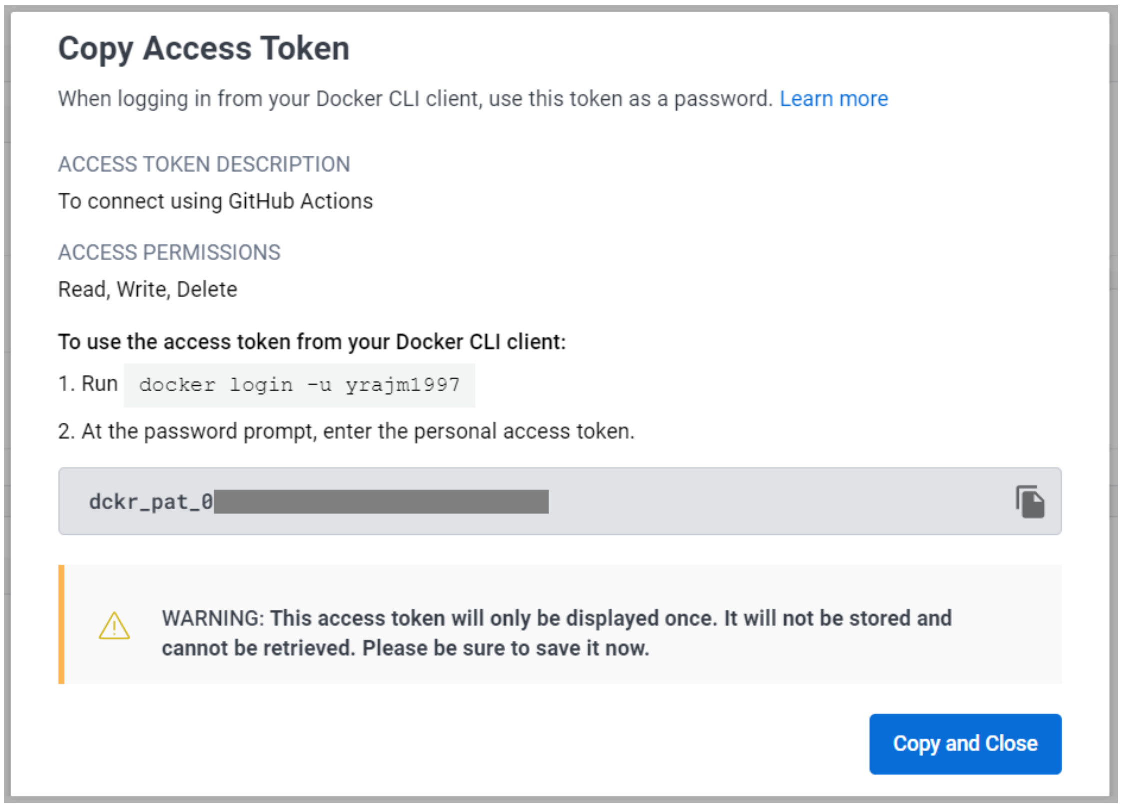
To connect using GitHub Actions

Access permissions

Read, Write, Delete

Read, Write, Delete tokens allow you to manage your repositories.

Access Token will be generated. Copy it somewhere safe, and close the window.


Copy Access Token

When logging in from your Docker CLI client, use this token as a password. Learn more

ACCESS TOKEN DESCRIPTION

To connect using GitHub Actions

ACCESS PERMISSIONS

Read, Write, Delete

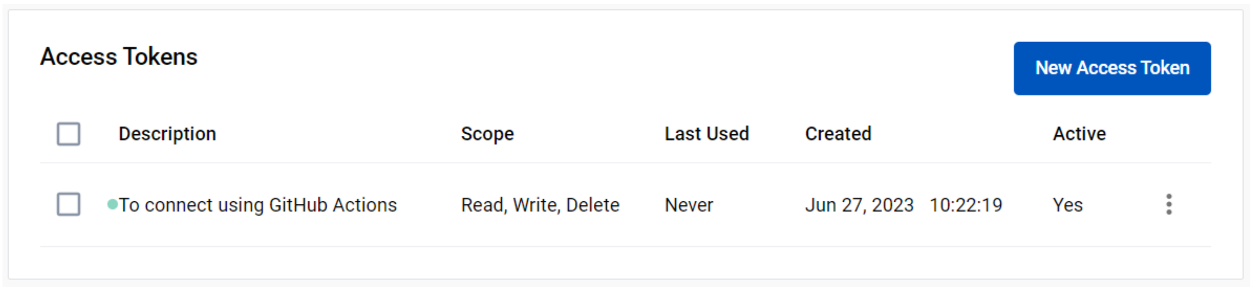
To use the access token from your Docker CLI client:

1. Run `docker login -u yrajm1997`
2. At the password prompt, enter the personal access token.

```
dckr_pat_0[redacted]
```

WARNING: This access token will only be displayed once. It will not be stored and cannot be retrieved. Please be sure to save it now.

You will see a new token added to the list.

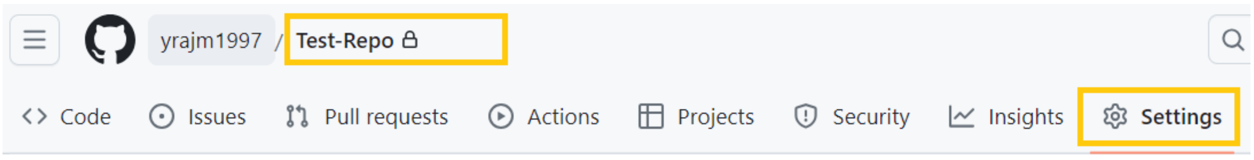


Access Tokens

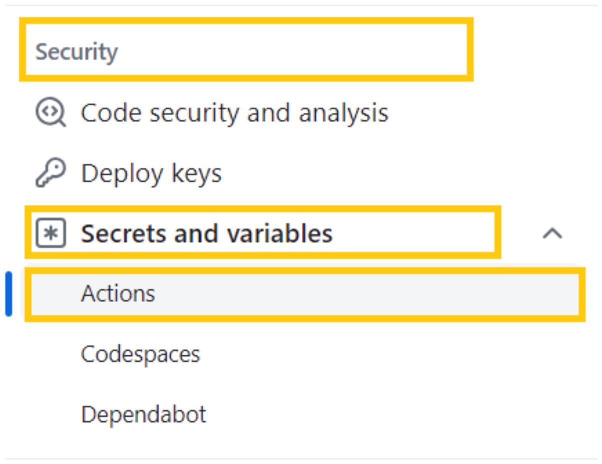
	Description	Scope	Last Used	Created	Active
☐	To connect using GitHub Actions	Read, Write, Delete	Never	Jun 27, 2023 10:22:19	Yes

7. Add your DockerHub account credentials (username, and passtoken) in GitHub secrets.

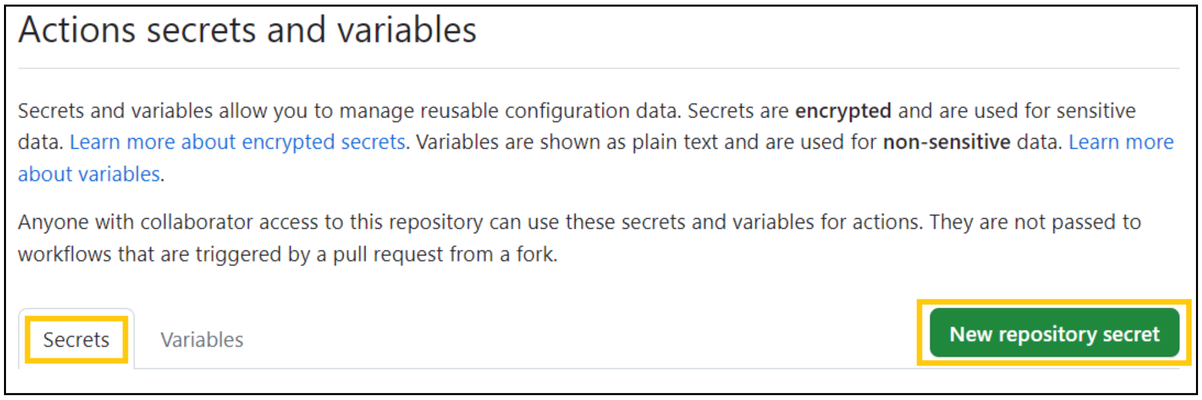
   Go to the Settings of your GitHub repository.



Select *Security > Secrets and variables > Actions* tab.



Select *Secrets > New repository secret*.

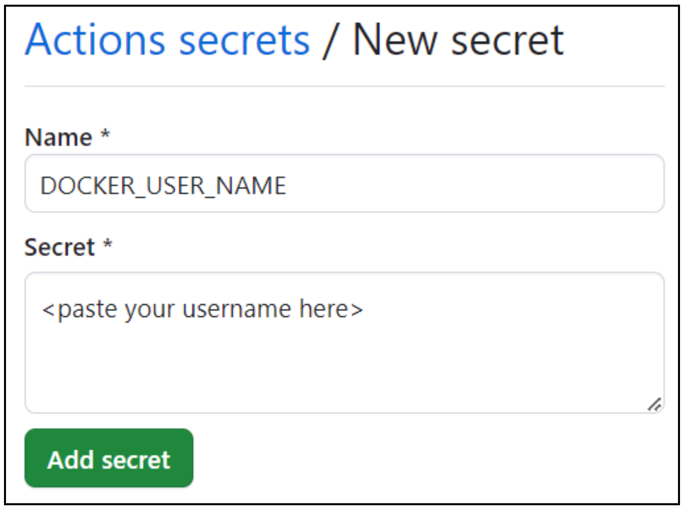


Actions secrets and variables

Secrets and variables allow you to manage reusable configuration data. Secrets are **encrypted** and are used for sensitive data. Learn more about encrypted secrets. Variables are shown as plain text and are used for **non-sensitive** data. Learn more about variables.

Anyone with collaborator access to this repository can use these secrets and variables for actions. They are not passed to workflows that are triggered by a pull request from a fork.

Give Name, and secret value, then click *Add secret*.



Actions secrets / New secret

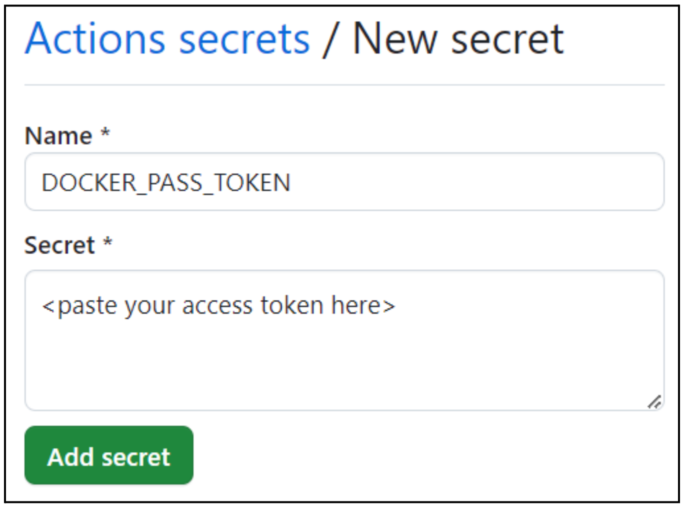
Name *

DOCKER_USER_NAME

Secret *

<paste your username here>

Similarly, add a secret for the access token.



Actions secrets / New secret

Name *

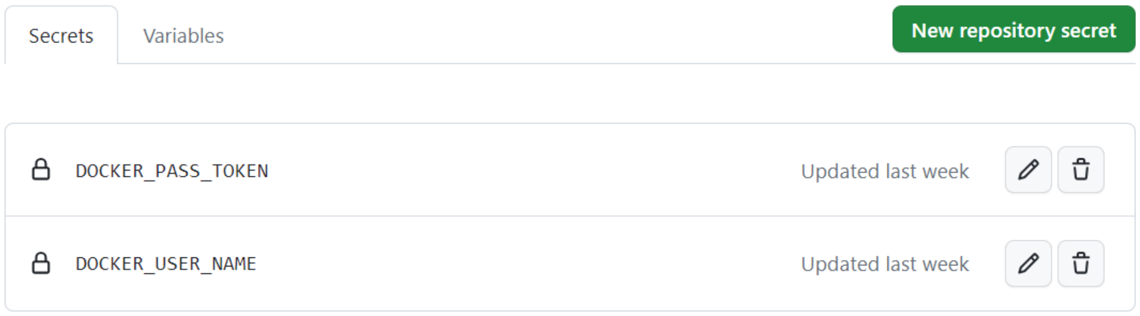
DOCKER_PASS_TOKEN

Secret *

<paste your access token here>

Secrets should be added to the list.

Note that these secrets will be accessed in the workflow yml file as `${{ secrets.DOCKER_USER_NAME }}`, and `${{ secrets.DOCKER_PASS_TOKEN }}`.

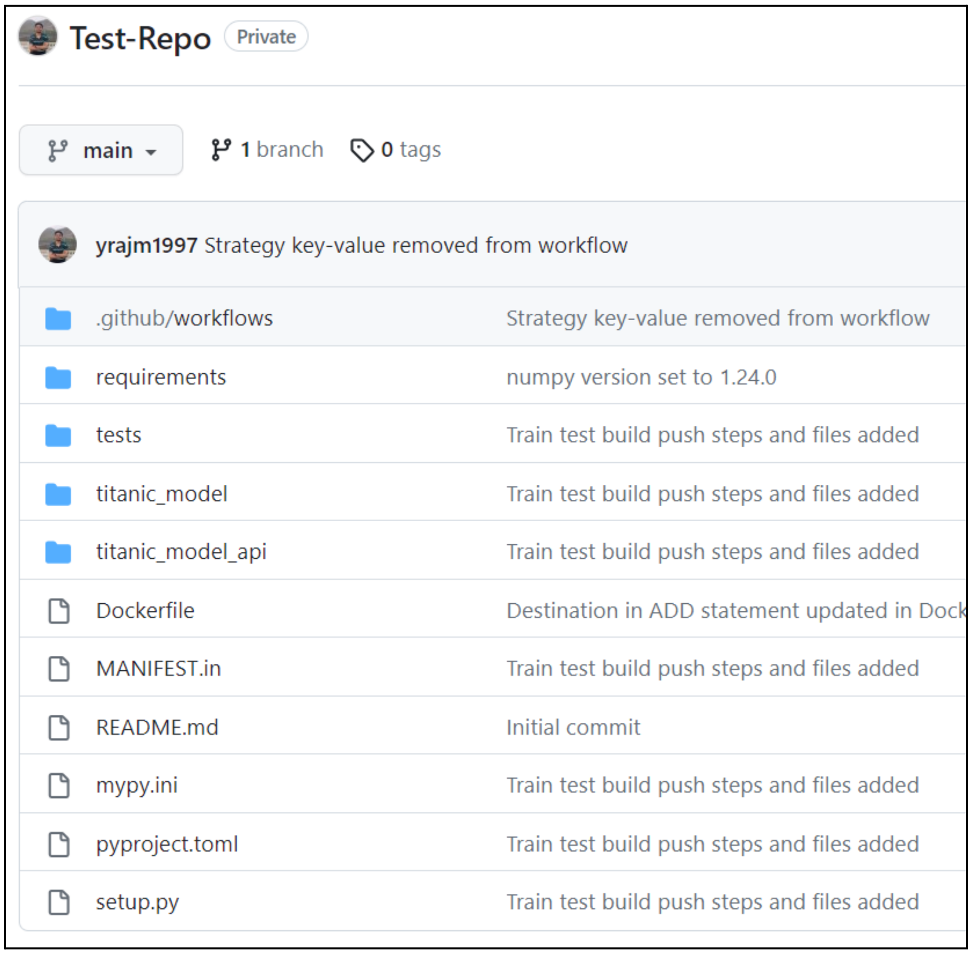


DOCKER_PASS_TOKEN	Updated last week
DOCKER_USER_NAME	Updated last week

8. Now, clone the GitHub repo either in your local system, or in a cloud environment.

   Add the content of the downloaded folder ‘GitHubActions_to_train_test_build_and_push’ to the cloned repo, and do a git push.

   Files should reflect on your GitHub repository page.



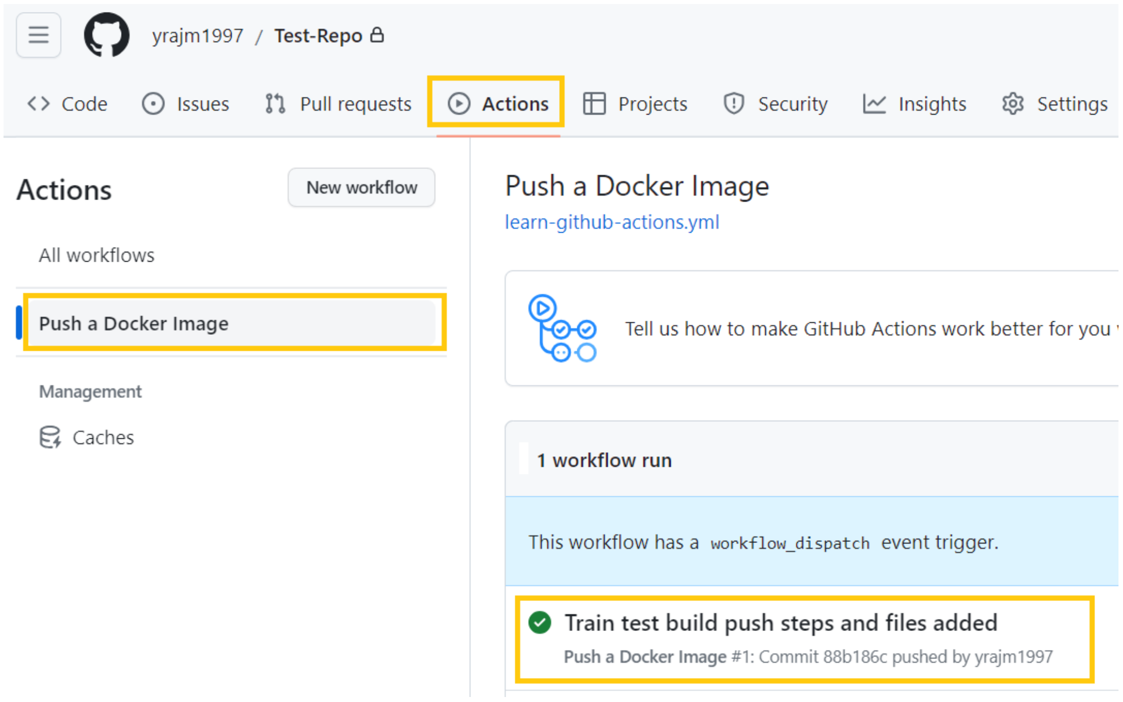
Test-Repo Private

main 1 branch 0 tags

yrajm1997 Strategy key-value removed from workflow

.github/workflows	Strategy key-value removed from workflow
requirements	numpy version set to 1.24.0
tests	Train test build push steps and files added
titanic_model	Train test build push steps and files added
titanic_model_api	Train test build push steps and files added
Dockerfile	Destination in ADD statement updated in Dock
MANIFEST.in	Train test build push steps and files added
README.md	Initial commit
mypy.ini	Train test build push steps and files added
pyproject.toml	Train test build push steps and files added
setup.py	Train test build push steps and files added

9. On the GitHub page of your repository, go to the Actions tab. Select your workflow run.



Push a Docker Image
learn-github-actions.yml

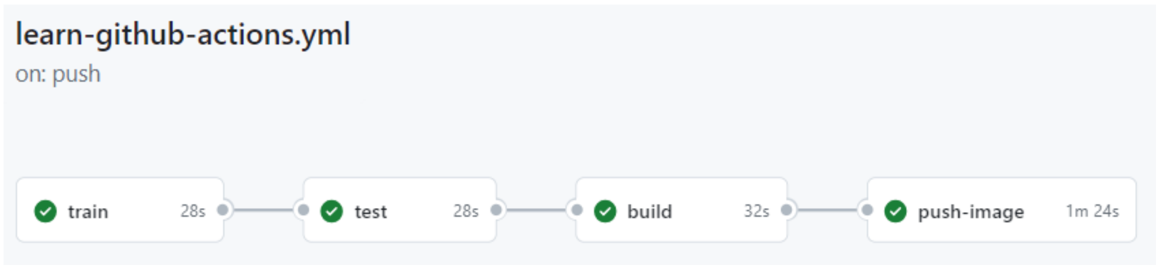
Tell us how to make GitHub Actions work better for you

1 workflow run

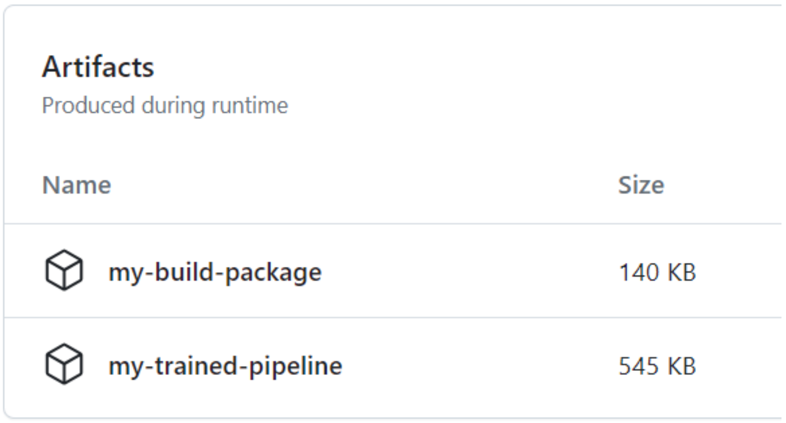
This workflow has a `workflow_dispatch` event trigger.

Train test build push steps and files added
Push a Docker Image #1: Commit 88b186c pushed by yrajm1997

10. Full pipeline should run successfully.



The artifacts that were uploaded during the workflow run will also be listed below the pipeline.



Artifacts
Produced during runtime

Name	Size
my-build-package	140 KB
my-trained-pipeline	545 KB