You have **2 free member-only stories left** this month. Sign up for Medium and get an extra one.

✨ Member-only story

GitHub Actions in MLOps: Automatically Check and Deploy Your ML Model

Automate Your ML Pipeline with GitHub Actions

Khuyen Tran · Follow
Published in Towards AI
8 min read · Jun 11, 2022

Listen Share

Motivation

Imagine your company is creating an ML-powered service. As a data scientist, you might try to continuously improve the existing ML model.

Once you find a better model, how do you make sure the service doesn't break when you deploy the new model?

Wouldn't it be nice if you can create a workflow that:

- Automatically tests a pull request from a team member
- Merges a pull request when all tests passed
- Deploys the ML model to the existing service?

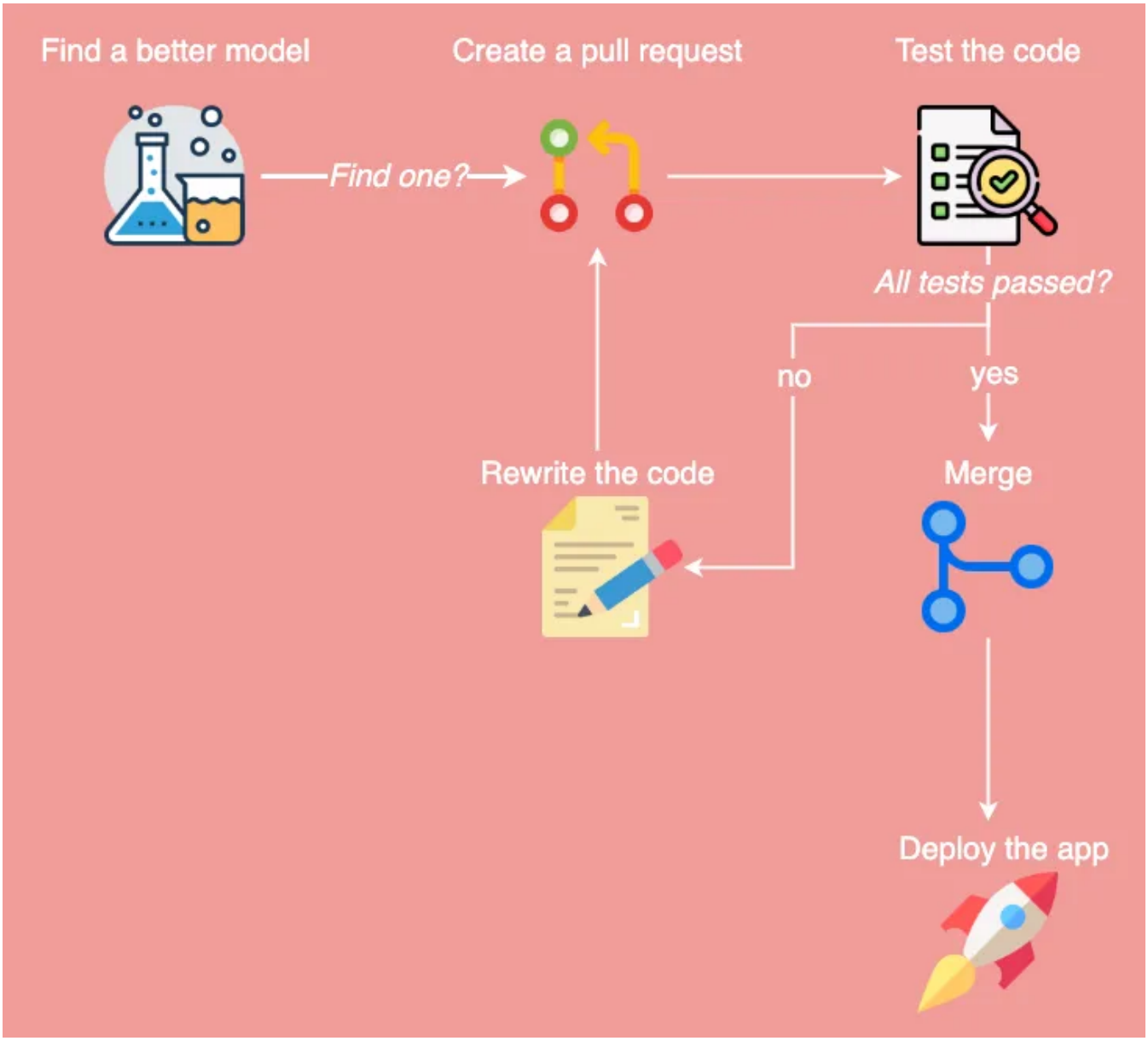


Image by Author

In this article, you will learn how to create such a workflow with GitHub Actions.

What are GitHub Actions?

GitHub Actions allows you to automate your workflows, making it faster to build, test, and deploy your code.

In general, a workflow will look similar to the below:

```yaml
name: Workflow Name # Name of the workflow
on: push # Define which events can cause the workflow to run
jobs: # Define a list of jobs
  first_job: # ID of the job
    name: First Job # Name of the job
    runs-on: ubuntu-latest # Name of machine to run the job on
    steps:
      ...
  second_job:
    name: Second Job
    runs-on: ubuntu-latest
    steps:
      ...
```

test_model.yaml hosted with ❤ by GitHub view raw

There are 3 important concepts to understand from the code above:

- When an *event* occurs (such as a push or a pull request), a *workflow* consisting of one or more *jobs* will be triggered
- *Jobs* are **independent** of each other. Each *job* is a set of *steps* that runs inside its own virtual machine runner or inside a container.
- *Steps* are **dependent** on each other and are executed in order.

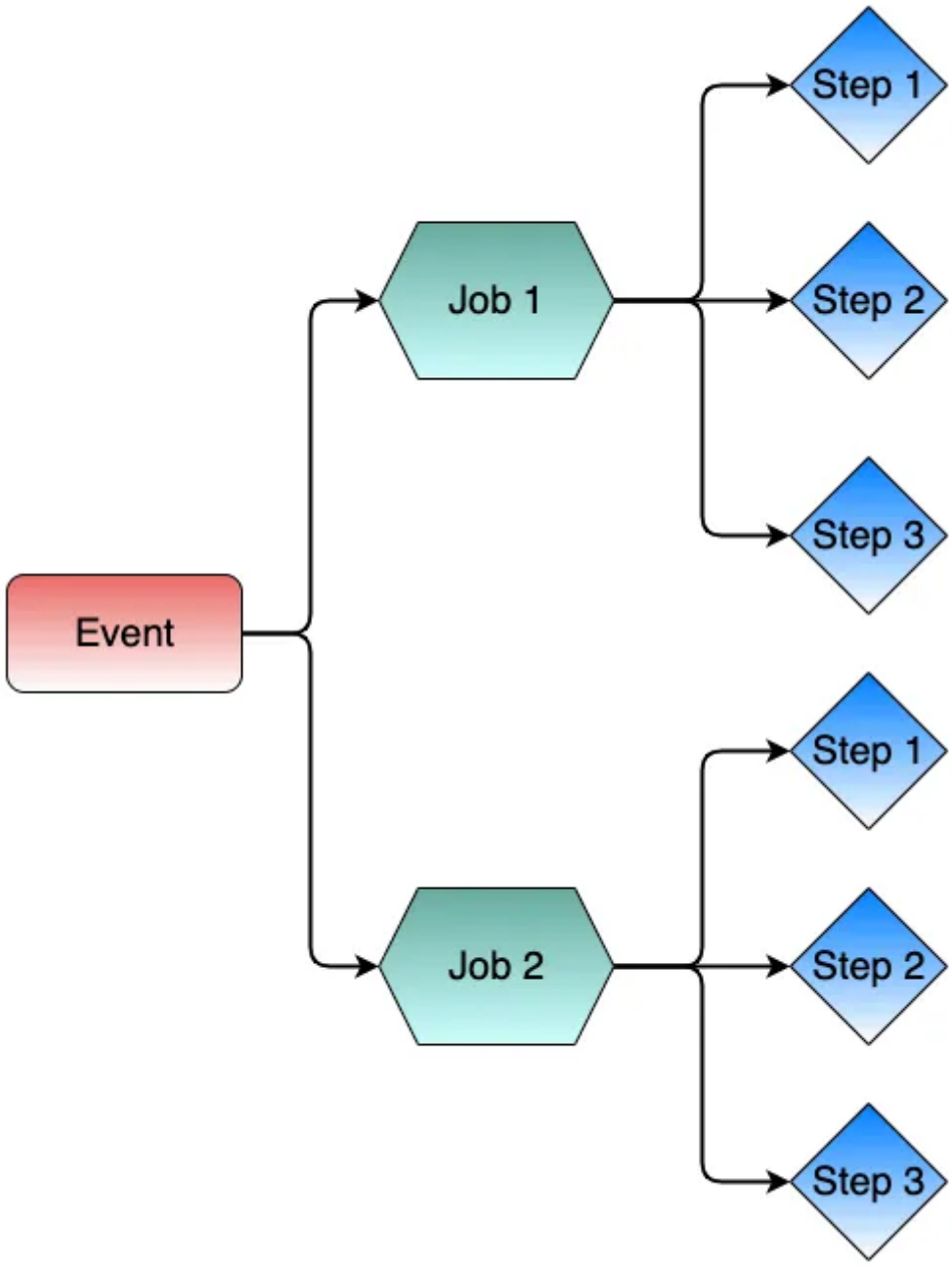


Image by Author

Let's dig deeper into these concepts in the next few sections.

Find the Best Parameters

The first steps in an ML project include experimenting with different parameters and models in a non-master branch. In the previous article, I mentioned how to use MLFlow + DagsHub to log your experiments.

DagsHub: a GitHub Supplement for Data Scientists and ML Engineers

Keep Your Data, Models, Experiments, and Code in One Place

towardsdatascience.com

Name	Commit	Cre... ↓	Sou...	use_label_en...	accuracy_sc...
fog parrotfish	a826	a minute ago	git	false	0.7239527389...
star lion	11b3	12 minutes ag	git	false	0.7067669172...

Image by Author

Link to the experiments shown above.

Once we found a combination of parameters and models that has a better performance than the existing model in production, we create a pull request to merge the new code with the master branch.

Use GitHub Actions to Test Code, ML Model, and Application

To make sure that merging new code won't cause any errors, we will create a workflow that:

- automatically tests the pull requests
- only allows the pull requests that pass all tests to merge with the master branch.

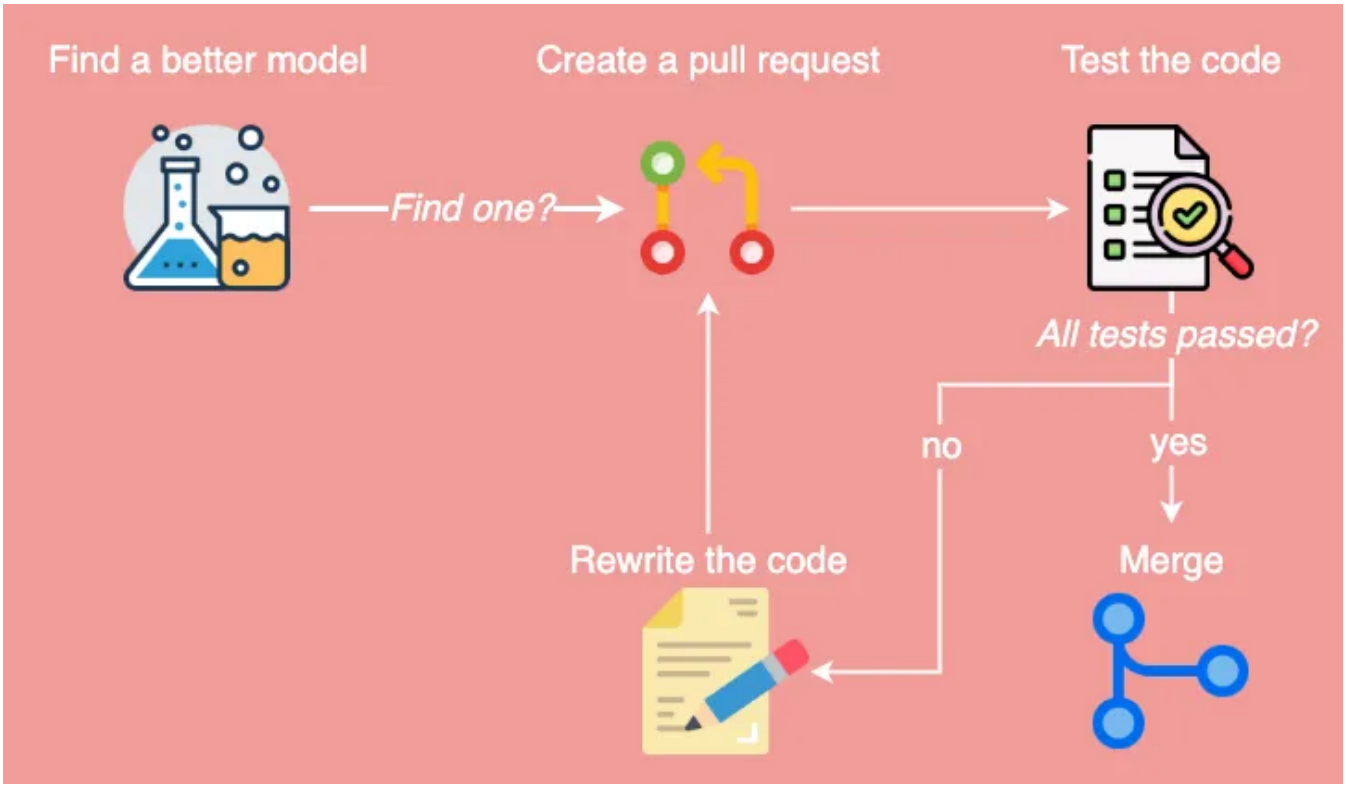


Image by Author

We will write this workflow inside a YAML file under `.github/workflows` .

```
.github
└── workflows
    └── test_code.yaml
```

Specify Events

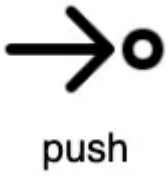


Image by Author

In this workflow, we use `on` to specify that the workflow will only run :

- If an event is a pull request.
- If the paths of the committed files match certain patterns.

```yaml
name: Test code and app
on:
  pull_request:
    paths: # Run when one or more paths match a pattern listed below
      - config/**
      - training/**
      - application/**
      - .github/workflows/test_code.yaml
```

test_code.yaml hosted with ❤ by GitHub view raw

Specify Steps

Next, create a job called `test_code` , which consists of several steps executed in order.

```yaml
jobs:
  test_model:
    name: Test new model
    runs-on: ubuntu-latest
    steps:
      ...
```

test_code.yaml hosted with ❤ by GitHub view raw

The first few steps will set up the environment before running the code.

```yaml
steps:
  - name: Checkout # Check out a repo
    uses: actions/checkout@v2

  - name: Environment setup # Set up with a specific version of Python
    uses: actions/setup-python@v2
    with:
      python-version: 3.8
      cache: pip

  - name: Cache # Cache dependencies
    uses: actions/cache@v2
    with:
      path: ~/.cache/pip
      key: ${{ runner.os }}-pip-${{ hashFiles('**/dev-requirements.txt') }}
      restore-keys: ${{ runner.os }}-pip-

  - name: Install packages # Install dependencies
    run: pip install -r dev-requirements.txt

  - name: Pull data # Get data from remote storage
    run: |
      dvc remote modify origin --local auth basic
      dvc remote modify origin --local user khuyentran1401
      dvc remote modify origin --local password MyScretPassword
      dvc pull -r origin train_model
```

test_code.yaml hosted with ❤ by GitHub view raw

Explanations of the syntax in the code above:

- `name` : A name for your step
- `uses` selects an *action,* which is an application that performs a complex but frequently repeated task. You can choose an action from thousands of actions

on GitHub Marketplace.

- `with` inserts input parameters required by an action
- `run` runs command-line programs using shell

Explanations of the steps:

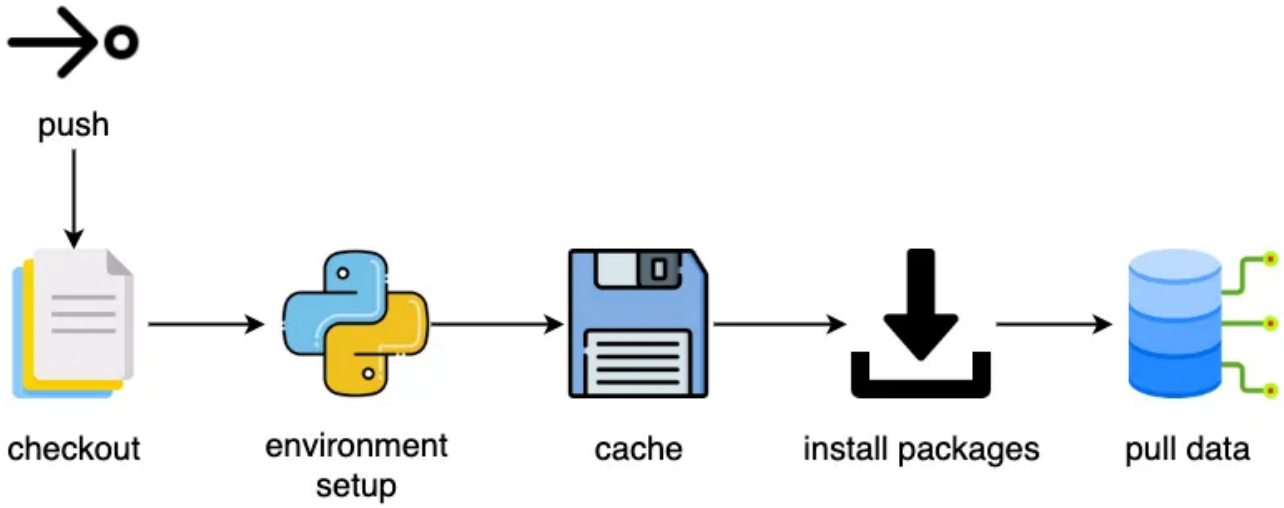


Image by Author

- `Checkout` checks out your repository so that the workflow can access files in your repository
- `Environment setup` sets up a Python environment for your workflow (I chose Python 3.8)
- `Cache` caches dependencies so that you don't need to install dependencies every time you run the workflow
- `Install packages` installs all dependencies your code needs to run successfully
- `Pull data` authenticates and pulls data from remote storage. Here, my remote storage is DagsHub

Note that it is risky to put your username and password in a script that everybody can see. Thus, we will use encrypted secrets to hide this confidential information.

Encrypted Secrets

Secrets are encrypted environment variables that you create in a repository. To create a secret, go to your repository, and click *Settings → Secrets → Actions → New repository secret*.

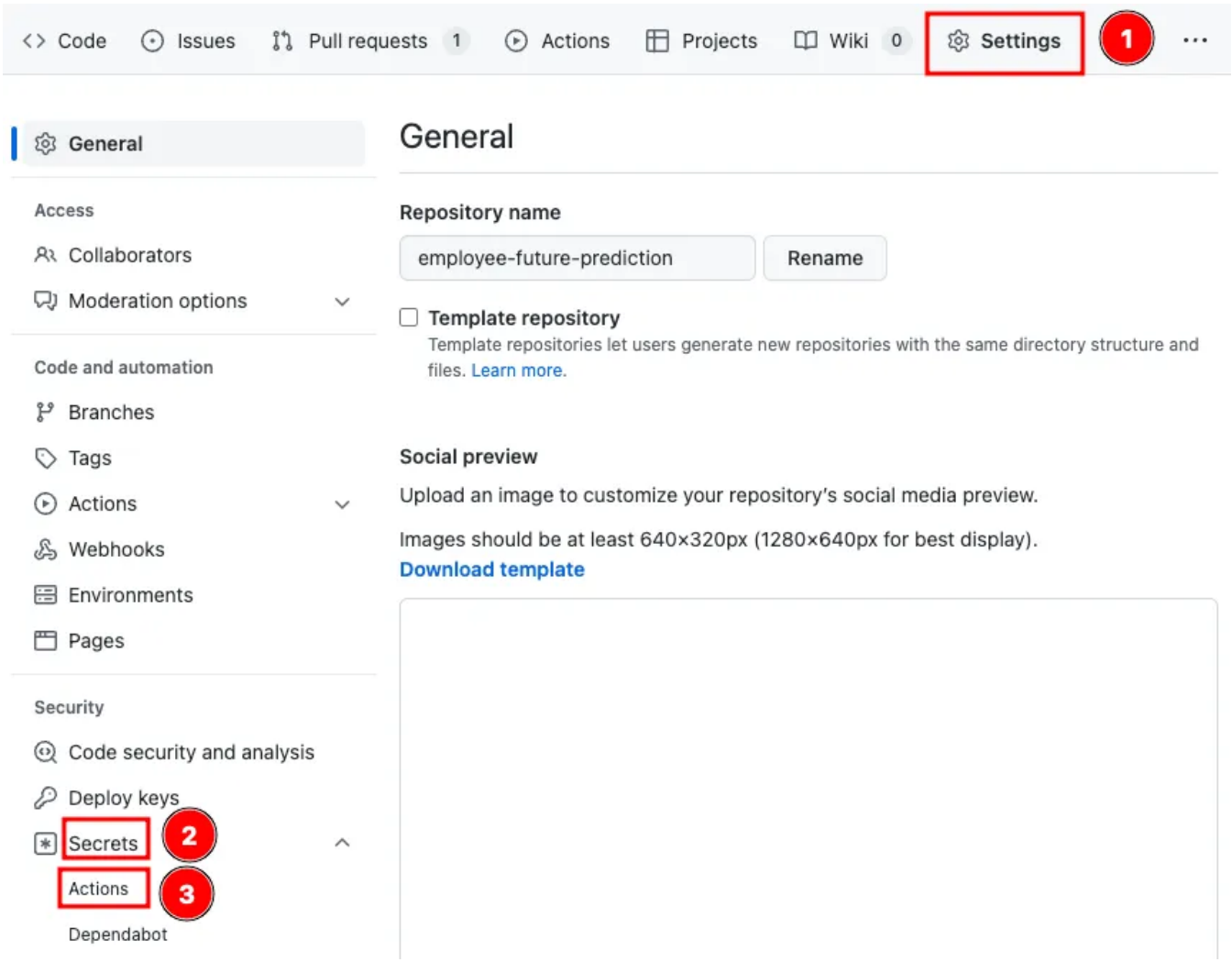


Image by Author

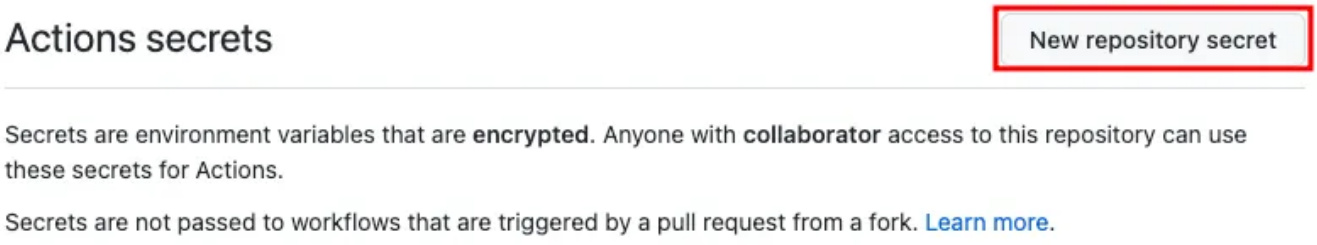


Image by Author

Insert the name of your secret and the value associated with this name.

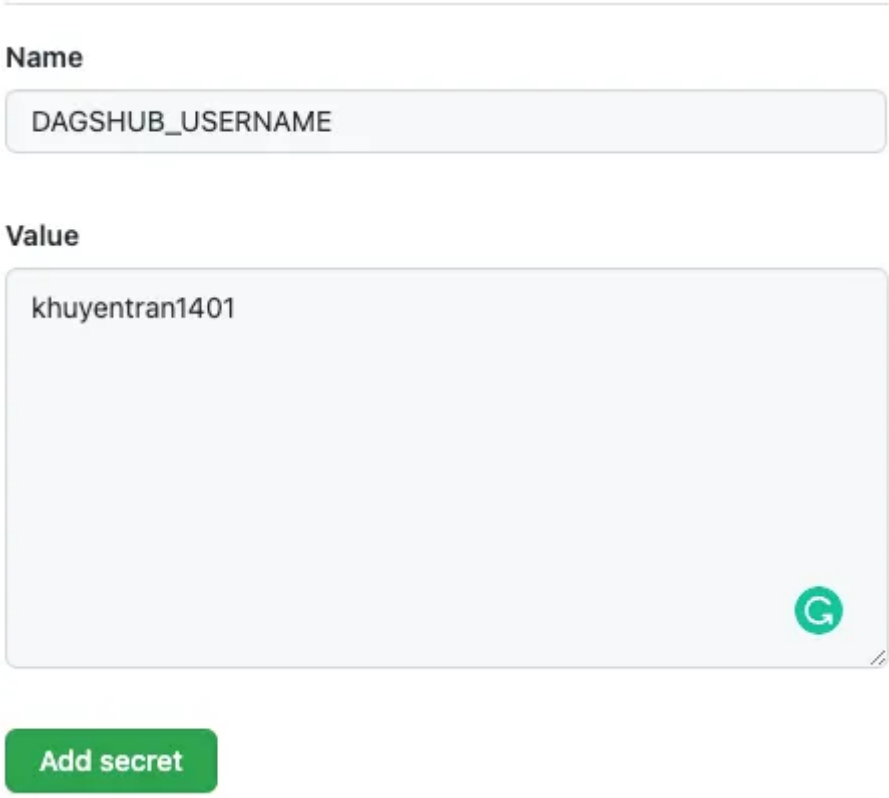


Image by Author

Now you can access the secret `DAGSHUB_USERNAME` using `${{ secrets.DAGSHUB_USERNAME }}` .

```yaml
steps:
  ...
  - name: Pull data
    run: |
      dvc remote modify origin --local auth basic
      dvc remote modify origin --local user ${{ secrets.DAGSHUB_USERNAME }}
      dvc remote modify origin --local password ${{ secrets.DAGSHUB_TOKEN }}
      dvc pull -r origin train_model
```

test_code.yaml hosted with ❤ by GitHub view raw

Run Tests

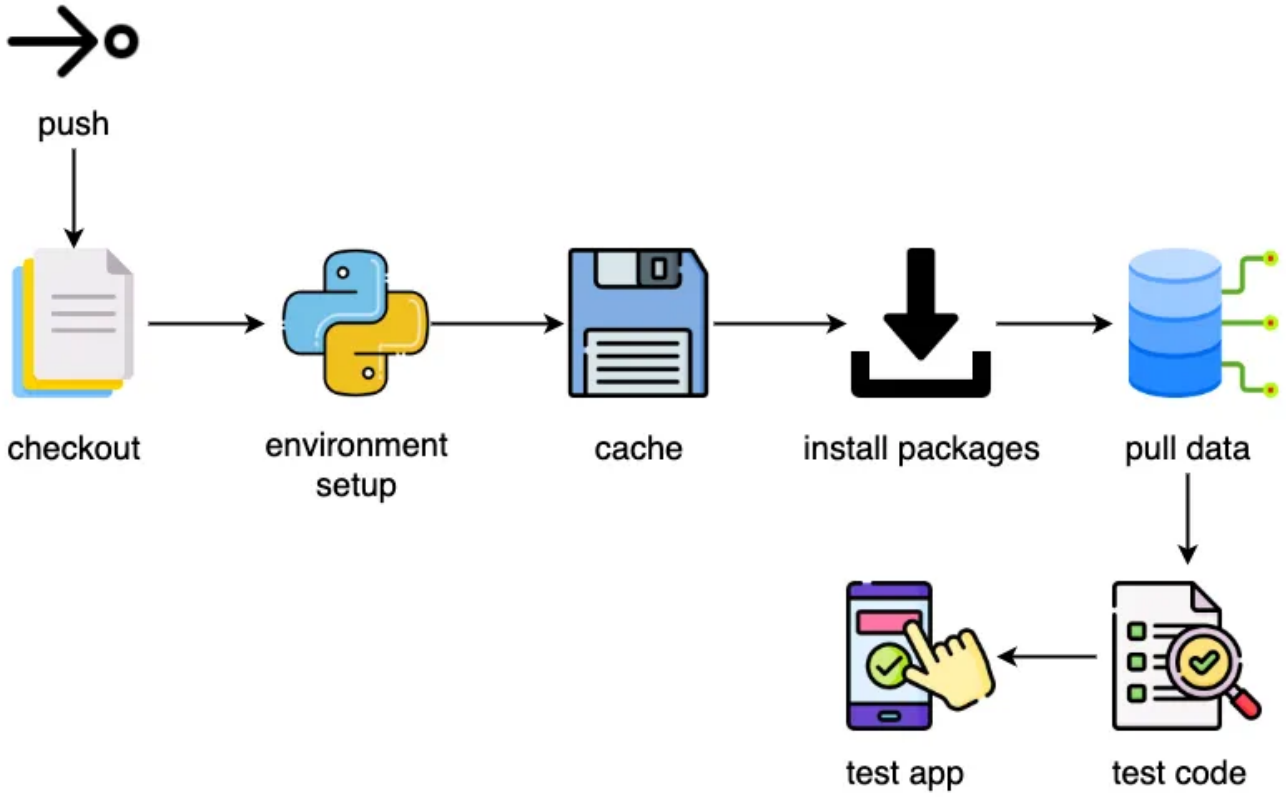


Image by Author

There are two parts to our code: training the model and deploying the model. We will write steps that make sure both parts can run without any errors and will work as expected.

Here is the step to test the training code:

```yaml
steps:
  ...
  - name: Run training tests
    run: pytest training/tests
```

test_code.yaml hosted with ❤ by GitHub view raw

Specifically, we test the processing code and ML model.

Find all the tests here.

The steps to test the deployment code include:

- Save a model to BentoML local store

```
steps:
  ...
  - name: Save model to BentoML local store
    run: python application/src/save_model_to_bentoml.py
```

test_code.yaml hosted with ❤ by GitHub view raw

- Run the application locally and run tests to make sure the application works as we expected.

```yaml
steps:
  ...
  - name: Serve the app locally and run app tests
    run: |
      bentoml serve ./application/src/create_service.py:service  &
      sleep 10
      pytest application/tests
      kill -9 `lsof -i:3000 -t`
```

test_code.yaml hosted with ❤ by GitHub view raw

Note: Here, we created an ML-powered app using BentoML. Read this article to understand more about BentoML:

BentoML: Create an ML Powered Prediction Service in Minutes

Containerize and Deploy Your ML Model in Python

towardsdatascience.com

Add and commit this workflow to the master branch on GitHub.

```
git add .github
git commit -m 'add workflow'
git push origin master
```

Add Rules

To make sure the code is available to be merged **only when** the workflow runs successfully, select *Settings* → *Branches* → *Add rule*.

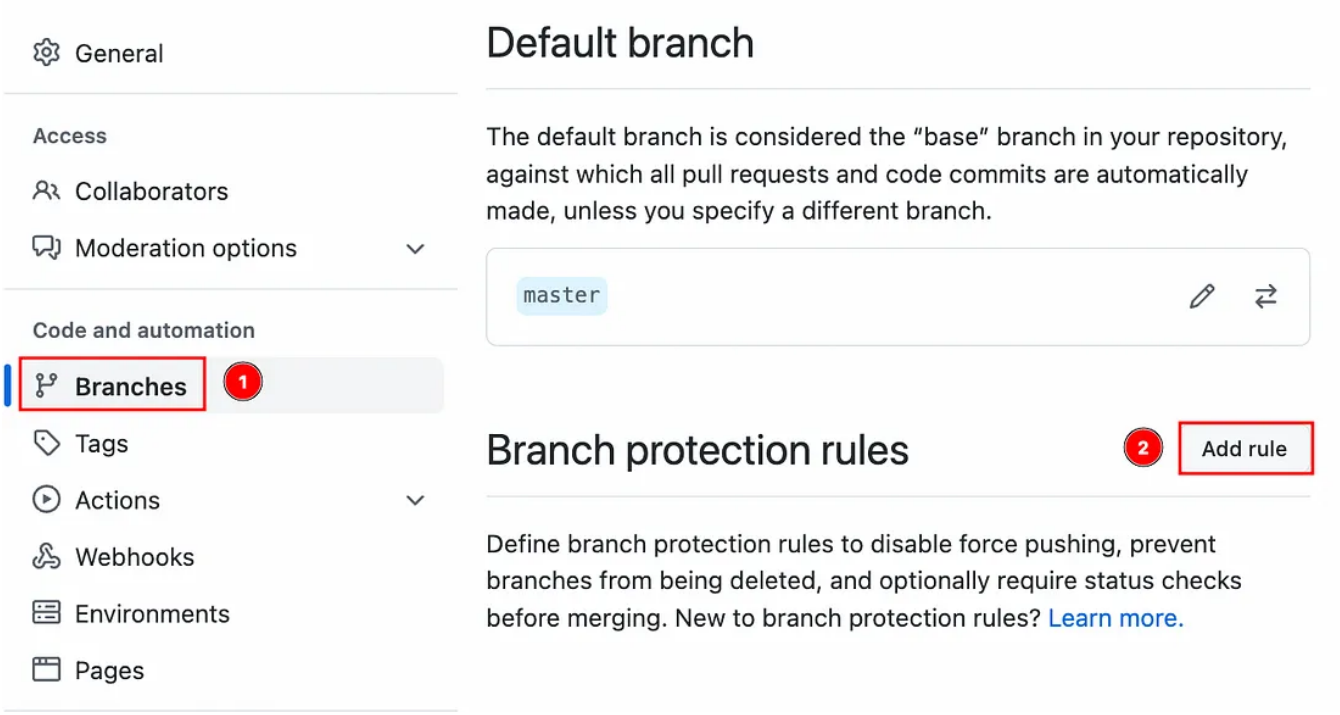


Image by Author

Add `master` as the branch name pattern, check `Require status checks to pass before merging` , then add the name of the workflow under *Status checks that are required*. Finally, click *Save changes*.

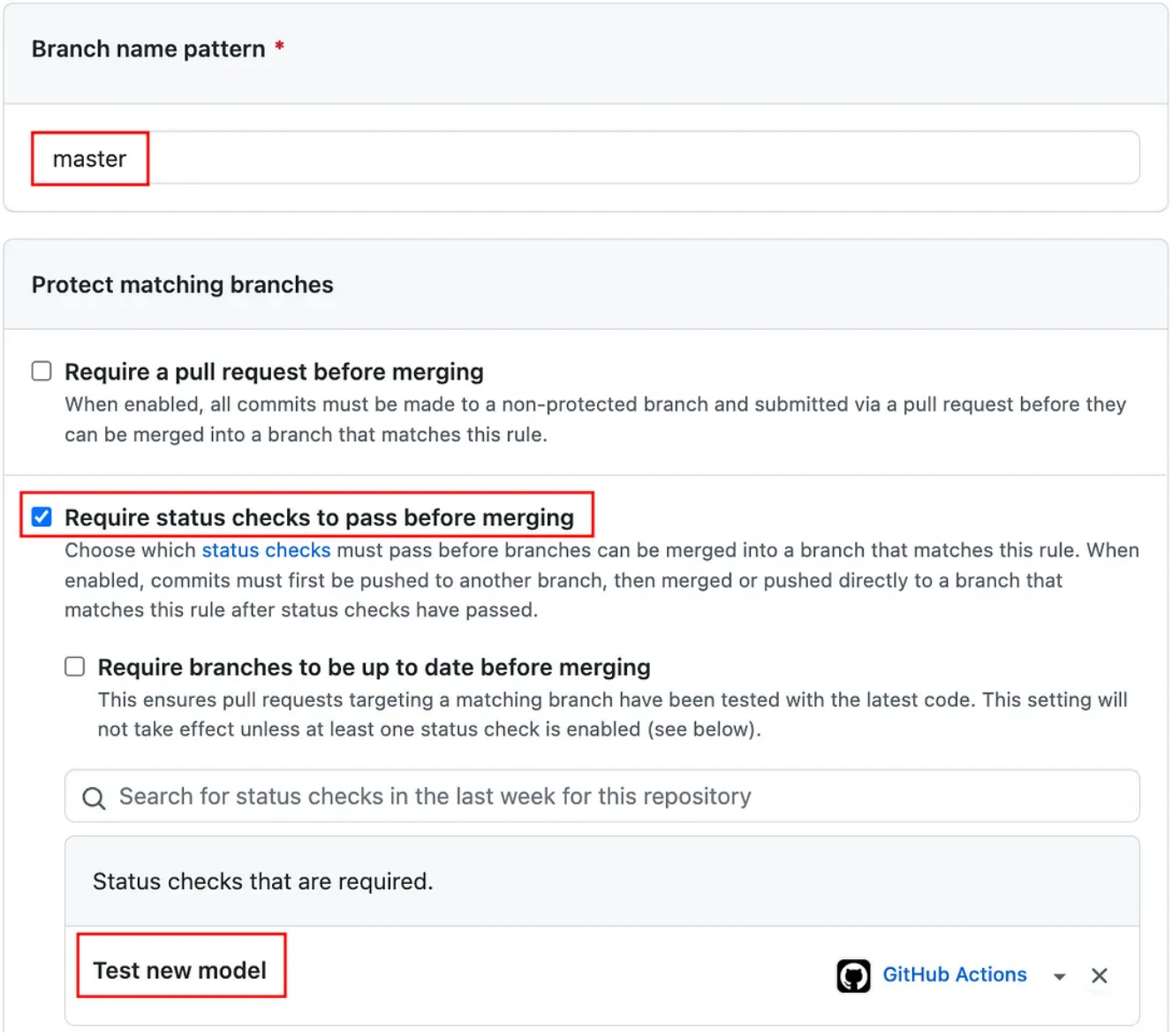


Image by Author

Now when you create a pull request, GitHub Actions will automatically run the workflow `Test new model`. You won't be able to merge the pull request if the check does not pass.

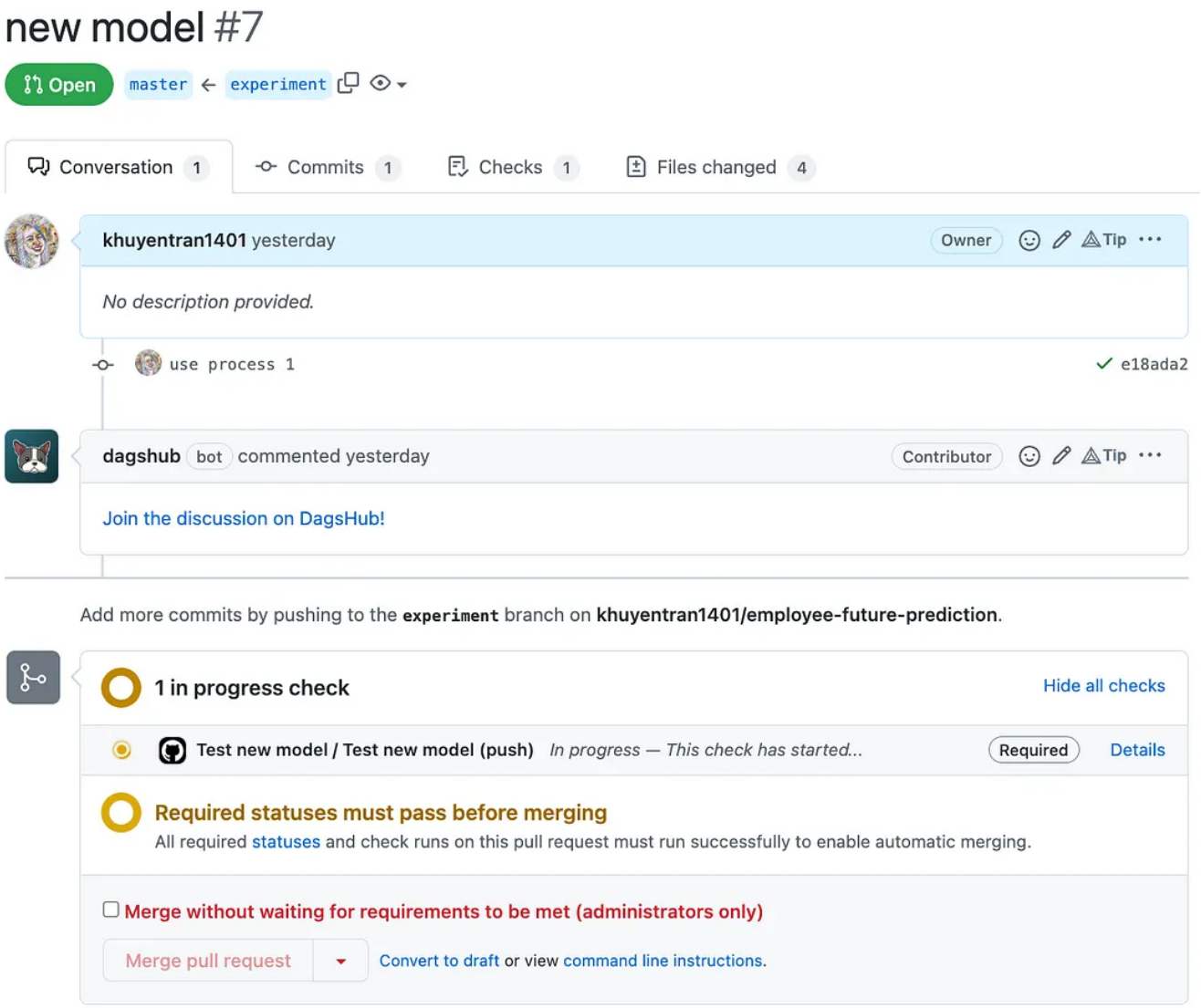


Image by Author

Clicking *Details* will show you the status of the run.

GIF by Author

Full code for testing the training code:

```yaml
name: Test new model
on:
  pull_request:
    paths:
      - config/**
      - training/**
      - application/**
      - .github/workflows/test_training.yaml
jobs:
  test_model:
    name: Test new model
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        id: checkout
        uses: actions/checkout@v2

      - name: Environment setup
        uses: actions/setup-python@v2
        with:
          python-version: 3.8
          cache: pip

      - name: Cache
        uses: actions/cache@v2
        with:
          path: ~/.cache/pip
          key: ${{ runner.os }}-pip-${{ hashFiles('**/dev-requirements.txt') }}
          restore-keys: ${{ runner.os }}-pip-

      - name: Install packages
        run: pip install -r dev-requirements.txt

      - name: Pull data
        run: |
          dvc remote modify origin --local auth basic
          dvc remote modify origin --local user ${{ secrets.DAGSHUB_USERNAME }}
          dvc remote modify origin --local password ${{ secrets.DAGSHUB_TOKEN }}
          dvc pull -r origin train_model

      - name: Run training tests
        run: pytest training/tests

      - name: Save model to BentoML local store
        run: python application/src/save_model_to_bentoml.py

      - name: Serve the app locally and run app tests
        run: |
```

```
49            bentoml serve ./application/src/create_service.py:service  &
50            sleep 10
51            pytest application/tests
52            kill -9 `lsof -i:3000 -t`
```

Use GitHub Actions to Deploy Model After Merging

After merging the pull request, the model should automatically be deployed to the existing service. Let's create a GitHub workflow to do exactly that.

Start with creating another workflow called `deploy_app.yaml` :

```
.github
└── workflows
    ├── deploy_app.yaml
    └── test_model.yaml
```

The first few steps of the workflow are similar to the previous workflow:

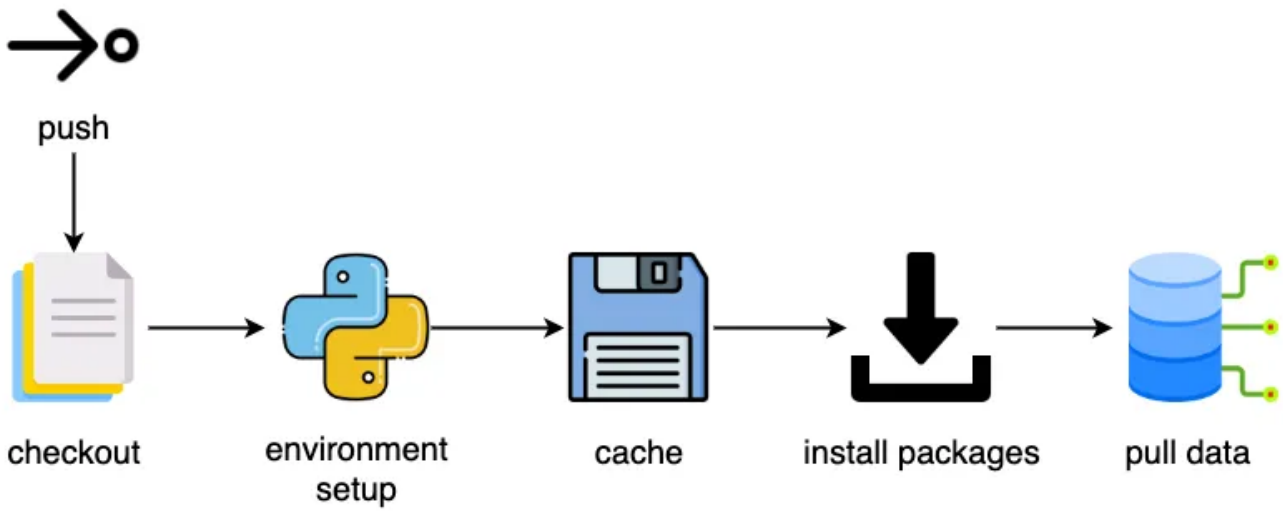


Image by Author

```yaml
name: Deploy App
on:
  push:
    branches:
      - master
    paths:
      - config/**
      - training/**
```

Open in app ↗ Sign up Sign In


```yaml
    name: Deploy App
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        id: checkout
        uses: actions/checkout@v2

      - name: Environment setup
        uses: actions/setup-python@v2
        with:
          python-version: 3.8
          cache: pip

      - name: Cache
        uses: actions/cache@v2
        with:
          path: ~/.cache/pip
          key: ${{ runner.os }}-pip-${{ hashFiles('**/dev-requirements.txt') }}
          restore-keys: ${{ runner.os }}-pip-

      - name: Install packages
        run: pip install -r dev-requirements.txt

      - name: Pull data
        run: |
          dvc remote modify origin --local auth basic
          dvc remote modify origin --local user ${{ secrets.DAGSHUB_USERNAME }}
          dvc remote modify origin --local password ${{ secrets.DAGSHUB_TOKEN }}
          dvc pull -r origin process_data train_model

      - name: Run and save model
        run: python training/src/evaluate_model.py
        env:
          MLFLOW_TRACKING_USERNAME: ${{ secrets.MLFLOW_TRACKING_USERNAME }}
          MLFLOW_TRACKING_PASSWORD: ${{ secrets.MLFLOW_TRACKING_PASSWORD }}
```

deploy_app.yaml hosted with ❤ by GitHub view raw

We also use `env` to add environment variables to the workflow. The environment variables will be used in some steps in the workflow.

```yaml
jobs:
  deploy_app:
    env: # Set environment variables
      HEROKU_API_KEY: ${{ secrets.HEROKU_API_KEY }}
      HEROKU_EMAIL: ${{ secrets.HEROKU_EMAIL }}
```

deploy_app.yaml hosted with ❤ by GitHub view raw

Next, we use BentoML to containerize the model and then deploy it to Heroku.

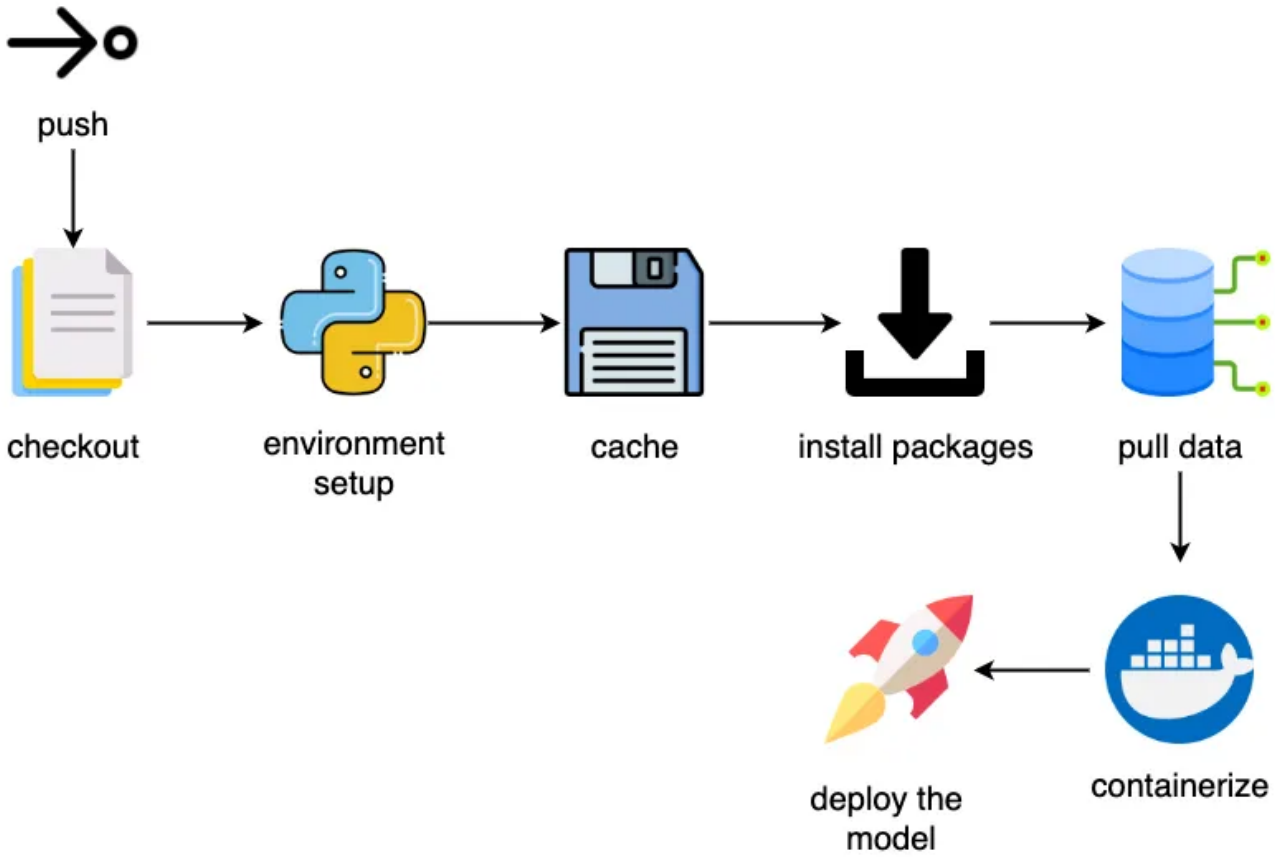


Image by Author

```yaml
steps:
  ...
  - name: Build Bentos
    run: bentoml build

  - name: Heroku login credentials
    run: |
      cat > ~/.netrc <<EOF
        machine api.heroku.com
          login $HEROKU_EMAIL
          password $HEROKU_API_KEY
        machine git.heroku.com
          login $HEROKU_EMAIL
          password $HEROKU_API_KEY
      EOF

  - name: Login to Heroku container
    run: heroku container:login

  - name: Containerize Bentos, push it to the Heroku app, and release the app
    run: |
      cd $(find ~/bentoml/bentos/predict_employee/ -type d -maxdepth 1 -mindepth 1)/env/docker
      APP_NAME=employee-predict-1
      heroku container:push web --app $APP_NAME  --context-path=../..
      heroku container:release web --app $APP_NAME
```

> *Full code for deploying the app.*

Add and commit this workflow to the master branch on GitHub.

```
git add .github
git commit -m 'add workflow'
git push origin master
```

Now when you merge a pull request, a workflow called `Deploy App` will run. To view the status of the workflow, click *Actions* → Name of the latest workflow → *Deploy App*.

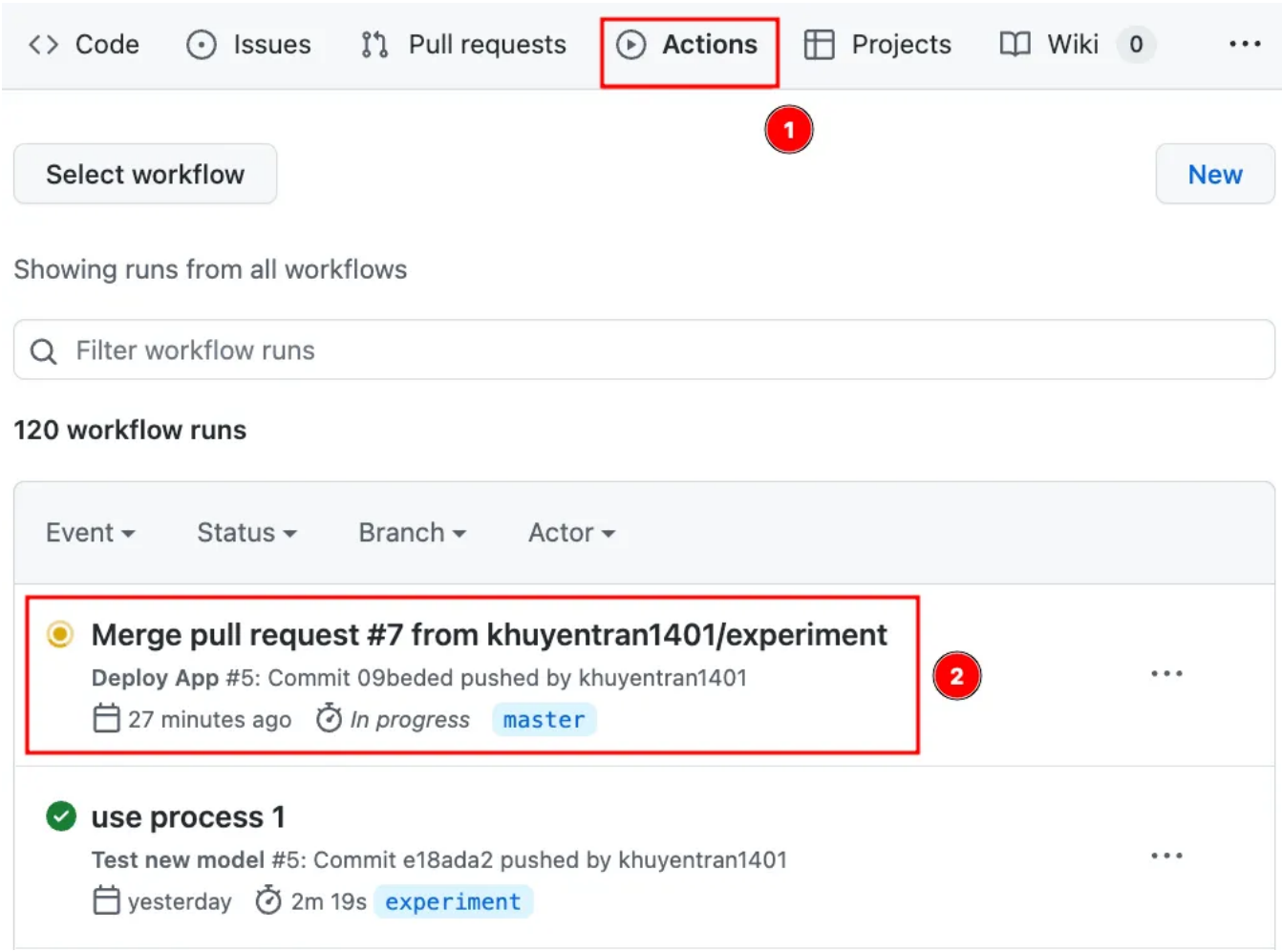


Image by Author

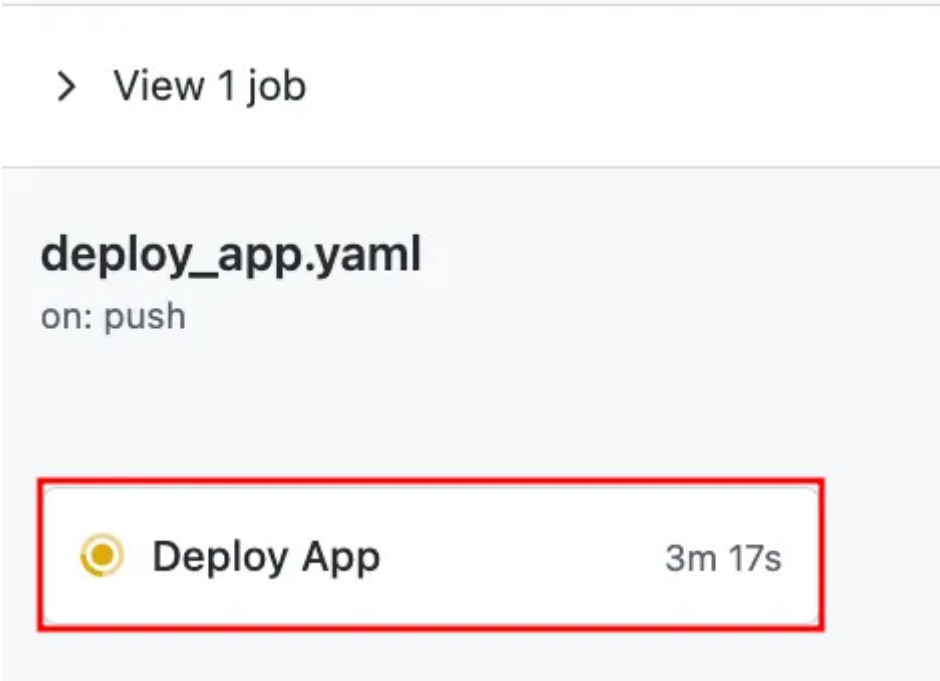
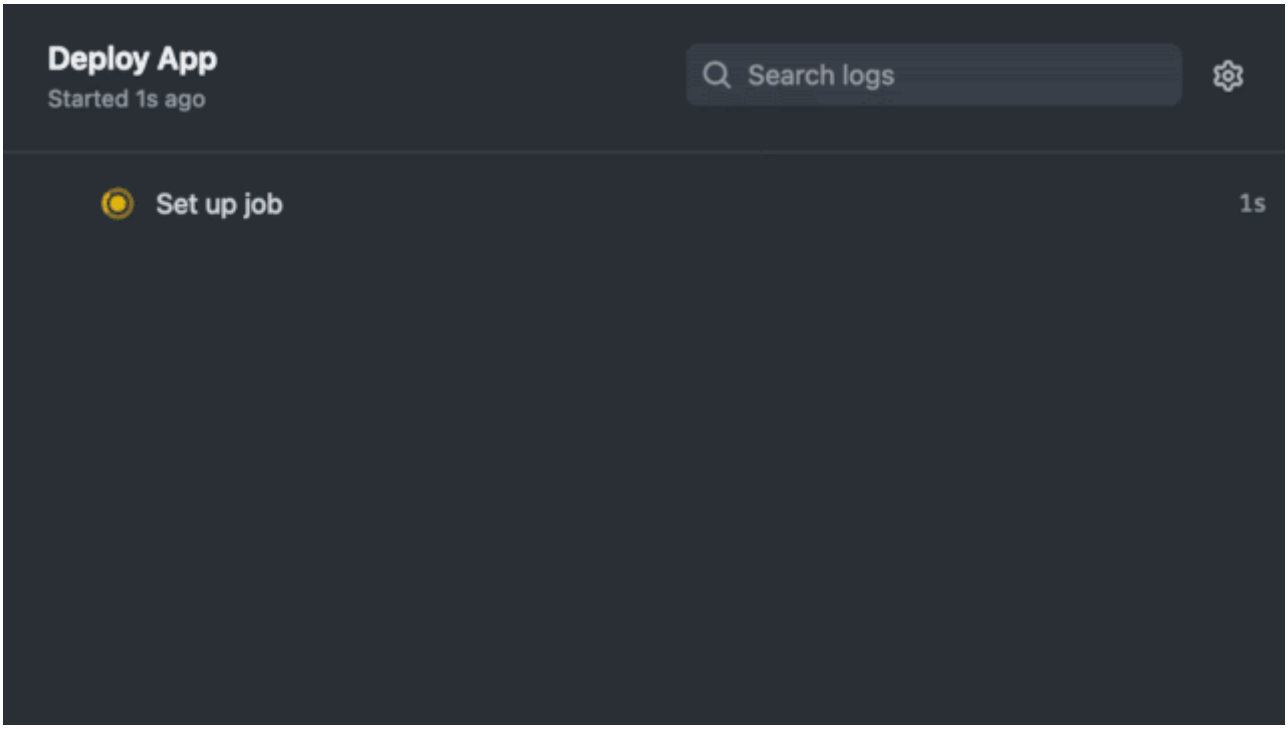


Image by Author

Now you should see your workflow running:



GIF by Author

Cool! The website for this app, which is https://employee-predict-1.herokuapp.com/, is now updated.

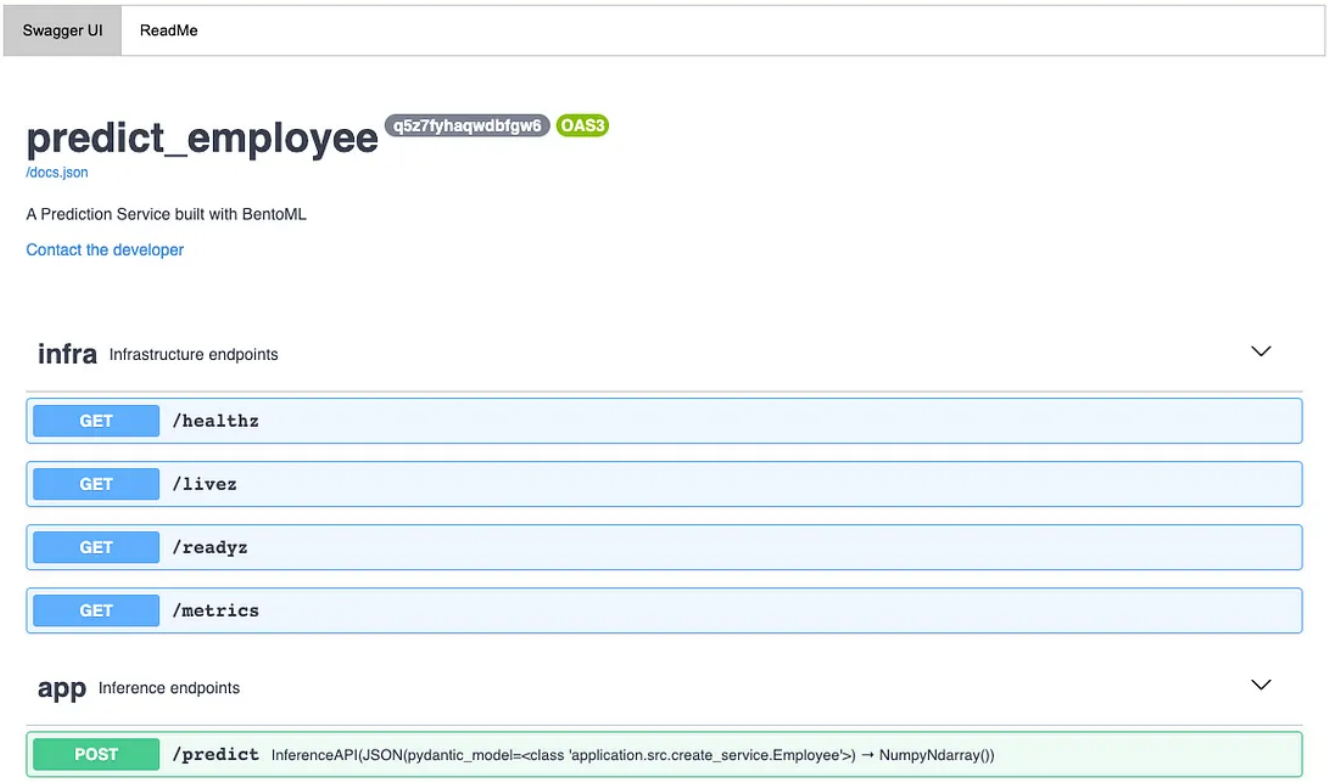
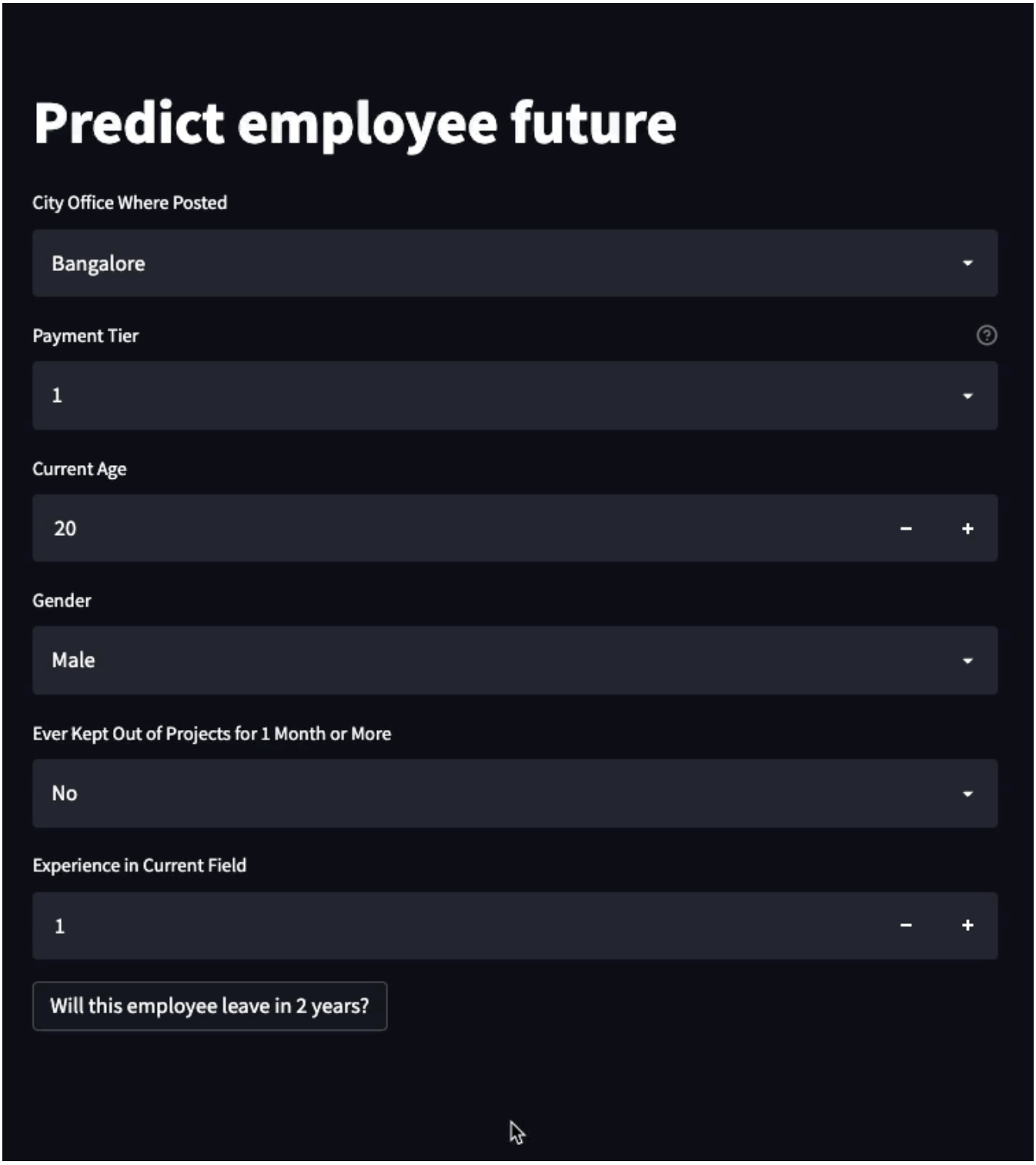


Image by Author

Since my Streamlit app makes the POST request to the URL above to generate predictions, the app is also updated.



GIF by Author

Conclusion

Congratulations! You have just learned how to use GitHub actions to create workflows that automatically test a pull request from a team member and deploy the ML model to the existing service. I hope this article will give you the motivation to automate your tasks with GitHub Actions.

Feel free to play and fork the source code of this article here:

GitHub - khuyentran1401/employee-future-prediction: Demo for Using GitHub Actions in MLOps

In a data science team, it is common to continuously try to find a better model than the existing one in production. It...

github.com

I like to write about basic data science concepts and play with different data science tools. You could connect with me on LinkedIn and Twitter.

Star this repo if you want to check out the codes for all of the articles I have written. Follow me on Medium to stay informed with my latest data science articles like these:

Introduction to DVC: Data Version Control Tool for Machine Learning Projects

Just like Git, but with Data!

towardsdatascience.com

BentoML: Create an ML Powered Prediction Service in Minutes

Containerize and Deploy Your ML Model in Python

towardsdatascience.com

DagsHub: a GitHub Supplement for Data Scientists and ML Engineers

Keep Your Data, Models, Experiments, and Code in One Place

towardsdatascience.com

Orchestrate a Data Science Project in Python With Prefect

Optimize Your Data Science Workflow in a Few Lines of Code

towardsdatascience.com

Reference

Deploy to Heroku with Github actions. remarkablemark. (2021, March 12). Retrieved May 31, 2022, from https://remarkablemark.org/blog/2021/03/12/github-actions-deploy-to-heroku/

Galvis, J. (2020, August 12). *Using Github actions for integration testing on a REST API*. Medium. Retrieved May 31, 2022, from https://medium.com/weekly-webtips/using-github-actions-for-integration-testing-on-a-rest-api-358991d54a20

Ktrnka. (n.d.). *Ktrnka/MLOPS_EXAMPLE_DVC: Mlops example using DVC, S3, and Heroku*. GitHub. Retrieved May 31, 2022, from https://github.com/ktrnka/mlops_example_dvc

Employee Future Prediction. CC0: Public Domain. Retrieved 2022–05–10 from https://www.kaggle.com/datasets/tejashvi14/employee-future-predictio

Mlops | Github Actions | Data Science | Machine Learning | Github

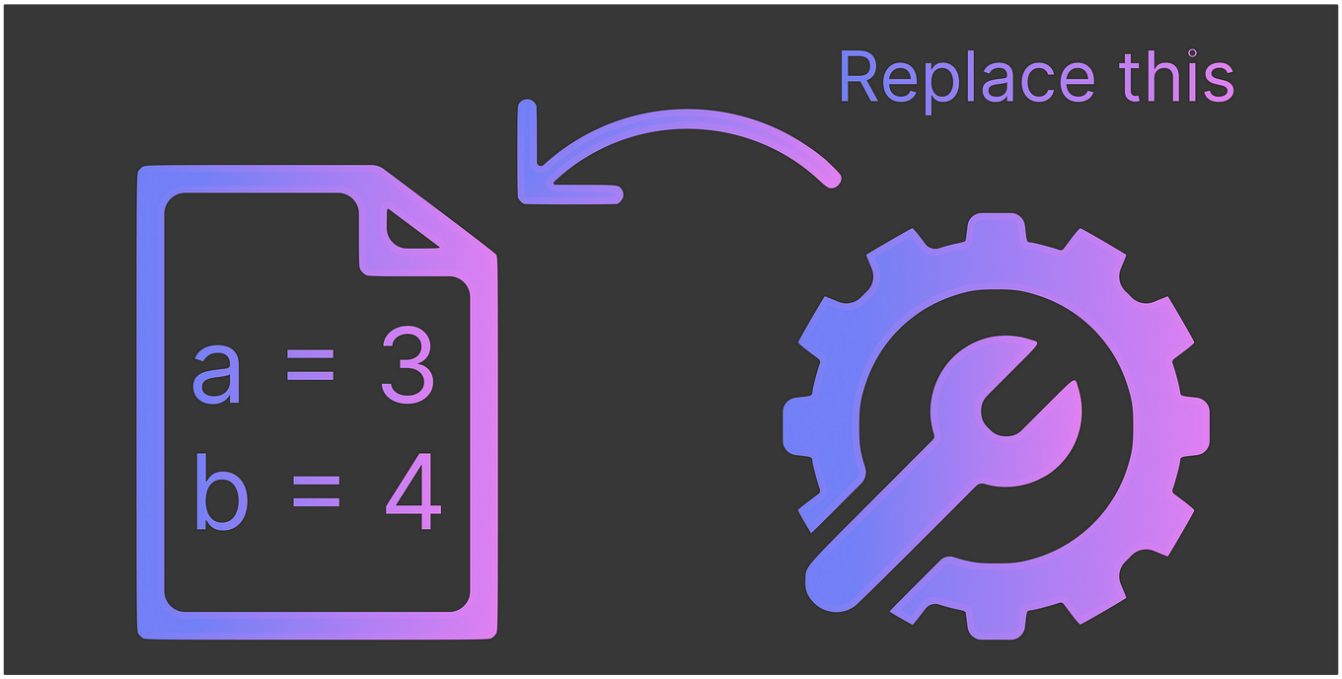
Follow

Written by Khuyen Tran

38K Followers · Writer for Towards AI

MLOps Engineer. Website: https://mathdatasimplified.com

More from Khuyen Tran and Towards AI



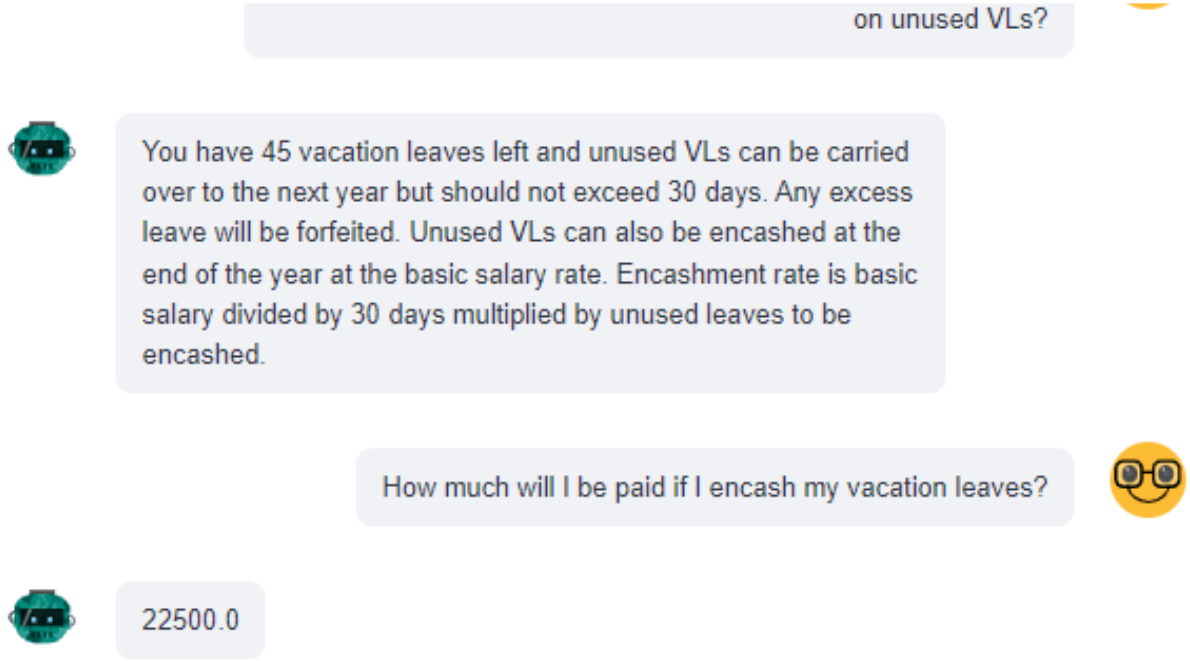
Khuyen Tran in Towards Data Science

Stop Hard Coding in a Data Science Project—Use Config Files Instead

And How to Efficiently Interact with Config Files in Python

· 6 min read · May 25

1.6K 20



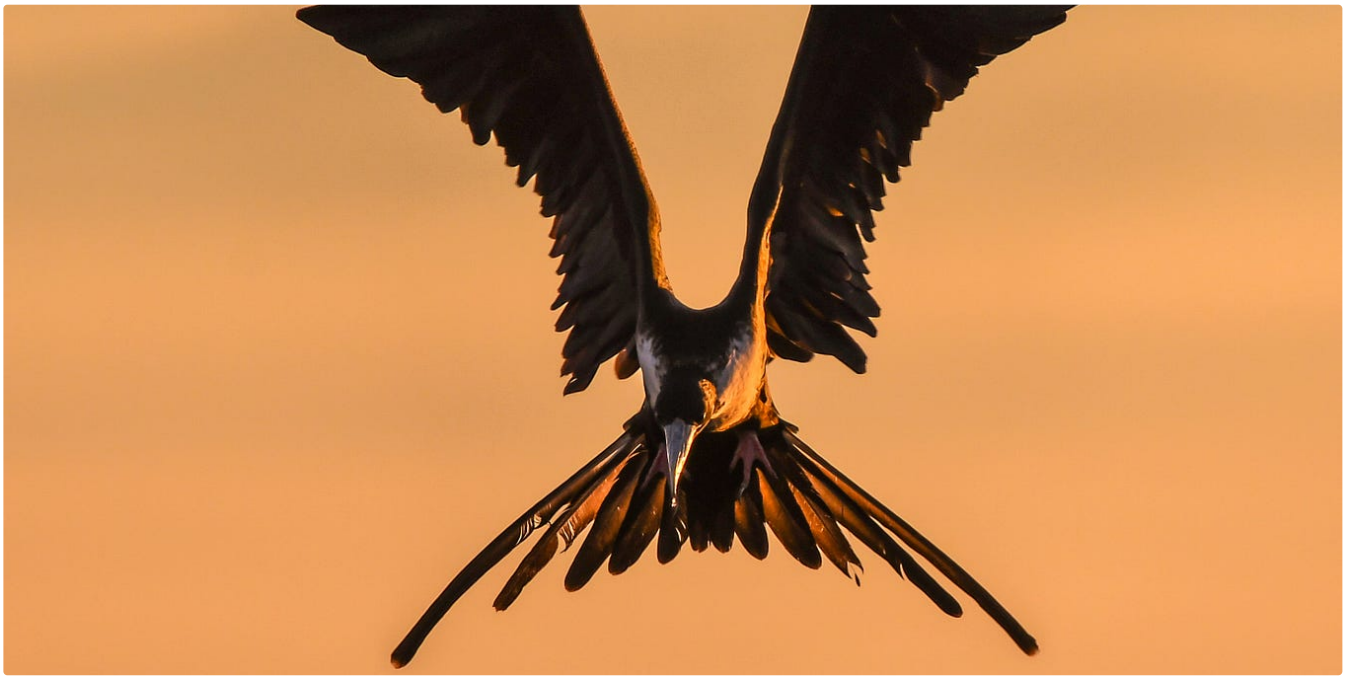
Stephen Bonifacio in Towards AI

Creating a (mostly) Autonomous HR Assistant with ChatGPT and LangChain's Agents and Tools

How to create a chatbot that can solve complex tasks autonomously using LLMs and LangChain.

13 min read · Jun 7

461 9



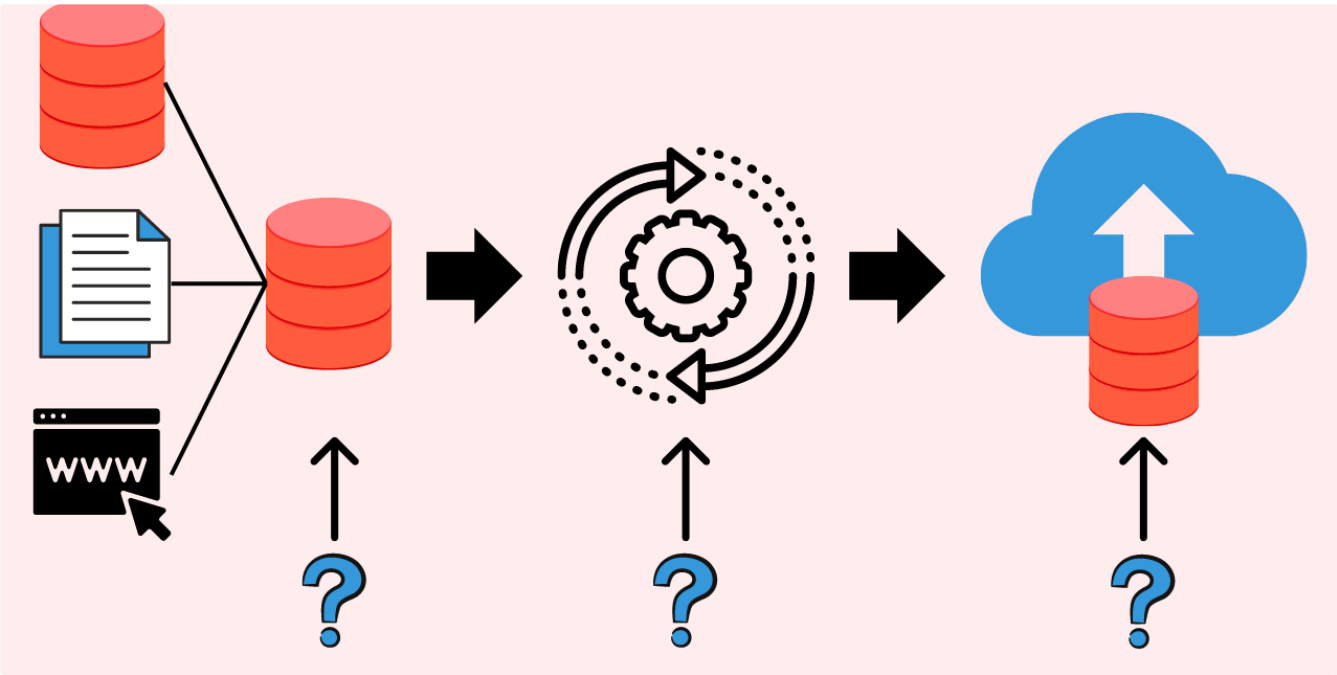
Dr. Mandar Karhade, MD. PhD. in Towards AI

Falcon-40B: A Fully OpenSourced Foundation LLM

Each Contributor hereby grants Grants to You a perpetual, worldwide, non-exclusive, irrevocable copyright license to reproduce, prepare...

· 7 min read · May 28

444 1



Khuyen Tran in Towards Data Science

What is dbt (data build tool) and When should you use it?

Discover the Hidden Benefits and Drawbacks of dbt

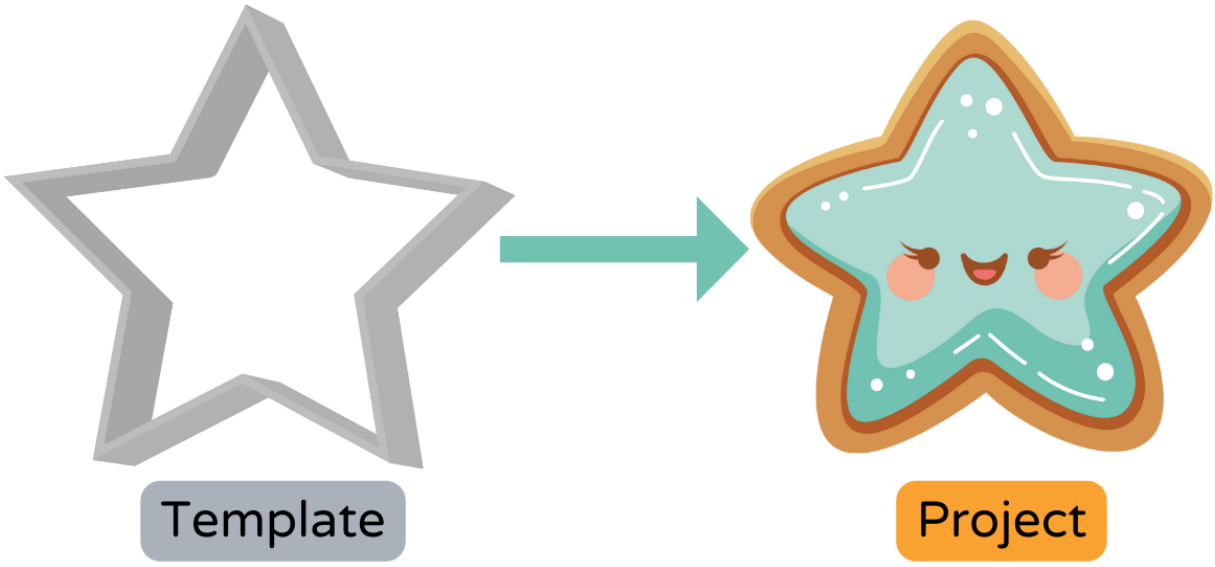
· 8 min read · Apr 30

477 1

See all from Khuyen Tran

See all from Towards AI

Recommended from Medium



Khuyen Tran in Towards Data Science

How to Structure an ML Project for Reproducibility and Maintainability

Start Your Next ML Project With This Template

· 7 min read · Jan 15

766 4

Bex T. in Towards AI

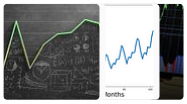
How To Create Highly-Organized ML Projects Anyone Can Reproduce With DVC Pipelines

What is a machine learning pipeline?

✨ · 11 min read · Jan 19

89 1

Lists

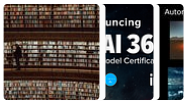


Predictive Modeling w/ Python

18 stories · 13 saves

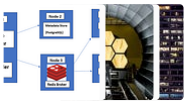
Practical Guides to Machine Learning

10 stories · 26 saves



Natural Language Processing

349 stories · 7 saves



New_Reading_List

173 stories · 1 save

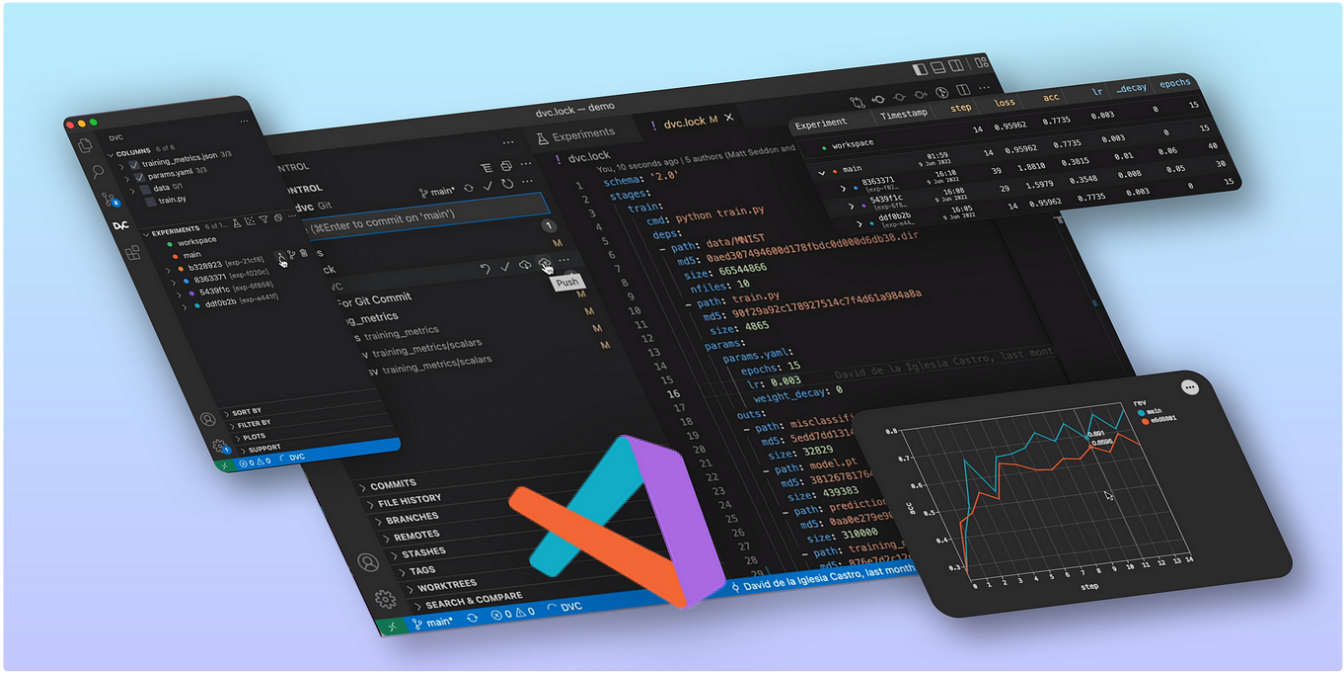
Bex T. in Towards Data Science

6 New Booming Data Science Libraries You Must Learn To Boost Your Skill Set in 2023

Data science isn't just Pandas, NumPy, and Scikit-learn anymore

· 7 min read · Jan 9

1.2K 6



Bex T. in Towards AI

How to Track ML Experiments With DVC Inside VSCode To Boost Your Productivity

Manage ML experiments like a pro

· 9 min read · Jan 17

226

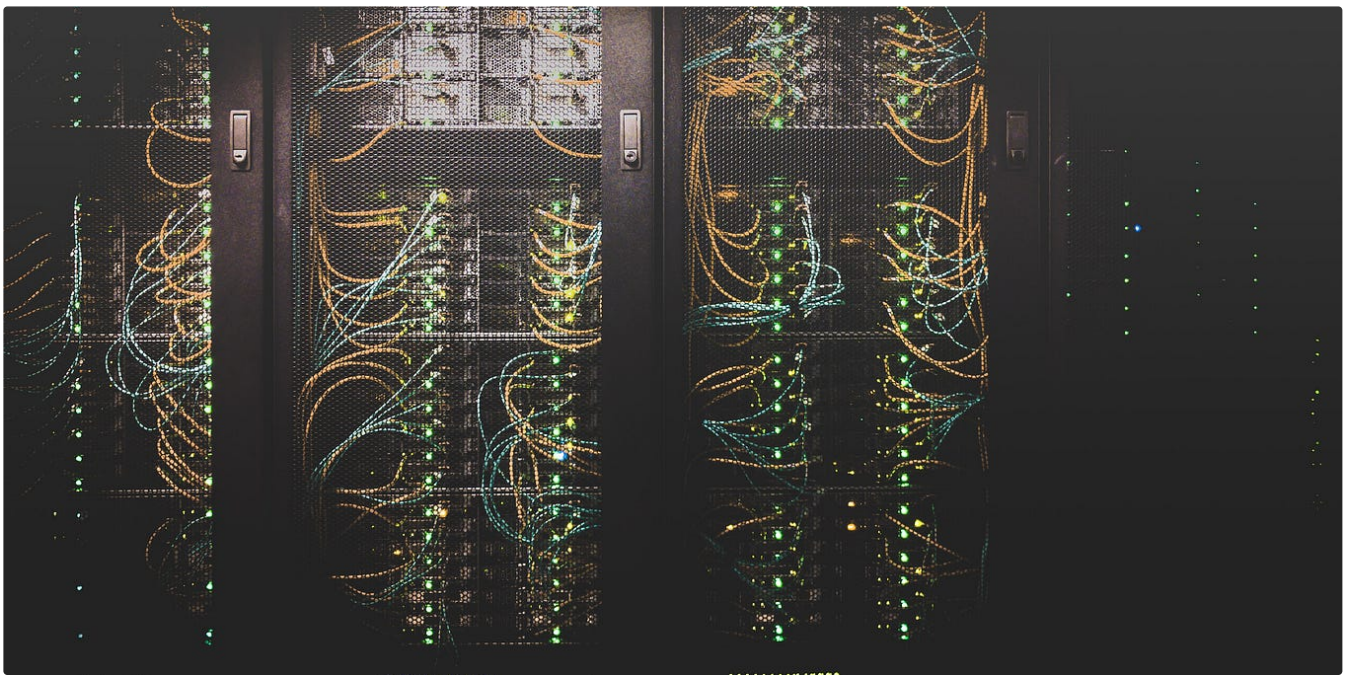
Bex T. in Towards AI

Bentoml vs. Fastapi: The Best ML Model Deployment Framework and Why It's Bentoml

Detailed comparison between BentoML and FastAPI for model deployment.

✨ · 9 min read · Jan 19

179 2



Barrett Studdard in Towards AI

Deploying ML Models with FastAPI and Azure

Learn how to deploy an API wrapper around a Machine Learning model

· 7 min read · Jan 13

44

See more recommendations