DevOps Activity

Advanced GitHub Actions for a Flask Application

Overview

Build and enhance a tiny Flask API, then wire up GitHub Actions for CI, linting, coverage reporting, security scans, and (optional) deployment. You'll finish with a repo that tests itself on every push/PR and tracks quality over time.

Learning outcomes

- Initialize a minimal Flask service and write a unit test.
- Run tests locally with pytest.
- Configure GitHub Actions for CI (test + coverage) and linting (Flake8).
- Add Codecov for coverage and Dependabot for dependency updates.
- Run CodeQL for basic security analysis.
- Use matrix builds to test across multiple Python versions.
- (Optional) Send Slack notifications and deploy to AWS.

Prerequisites

- Python 3.10+ installed locally
- Git + GitHub account
- (Optional) AWS account/credentials for deployment

What you will submit

- ZIP file with the project files
- Screenshot(s) of passing GitHub Actions runs (tests, lint, CodeQL)
- A short README section describing what you implemented (endpoints, tests) and any issues you hit

Part 1 — Bootstrap the Flask app

1. Create a project folder and a virtual environment, then install Flask.

```shell
python3 -m venv venv
source venv/bin/activate
pip install flask pytest
```

2. Create requirements.txt.

```shell
pip freeze > requirements.txt
```

3. Add .gitignore with:

```shell
venv/
__pycache__/
*.pyc
.DS_Store
```

4. Create app.py.

```python
# app.py
from flask import Flask, jsonify, request

app = Flask(__name__)

@app.route('/hello', methods=['GET'])
def hello():
    return jsonify(message="Hello, World!")

if __name__ == '__main__':
    app.run(debug=True)
```

5. Create test_app.py.

Python

```python
# test_app.py
import app

def test_hello():
    client = app.app.test_client()
    response = client.get('/hello')
    assert response.status_code == 200
    assert response.get_json() == {"message": "Hello, World!"}
```

6. Run tests locally.

Shell

```
pytest -q
```

Part 2 — Push to GitHub & add CI (tests)

1. Initialize Git and push.

Shell

```
git init
git add .
git commit -m "Initial commit"
git branch -M main
git remote add origin
https://github.com/YOUR_USERNAME/YOUR_REPO_NAME.git
git push -u origin main
```

2. Create the test workflow: .github/workflows/test.yml (you might need to tweak some parameters)

None

```
name: Python Application Test

on: [push, pull_request]
```

```
jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.10'

      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt

      - name: Run tests with coverage
        run: pytest --cov=app test_app.py
```

3. Commit and push the workflow. Open Actions in GitHub and confirm it runs.

If the runner can’t find pytest or pytest-cov, ensure they are in requirements.txt.

Part 3 — Linting with Flake8

1. Install Flake8 locally and try it.

Shell

```
pip install flake8
flake8 app.py test_app.py
```

2. Add a lint workflow: .github/workflows/lint.yml

None

```
name: Python Linting

on: [push, pull_request]

jobs:
  lint:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.10'

      - name: Install Flake8
        run: pip install flake8

      - name: Run Flake8
        run: flake8 app.py test_app.py
```

Part 4 — Coverage reporting with Codecov

1. Ensure pytest-cov is installed (locally and in CI).

Shell

```
pip install pytest-cov
pip freeze > requirements.txt
```

2. Add Codecov upload to your test workflow (after tests):

```
None
- name: Upload coverage to Codecov
  uses: codecov/codecov-action@v2
  with:
    token: ${{ secrets.CODECOV_TOKEN }}  # Add in Repo Settings → Secrets and variables → Actions
```

3. Push and verify coverage appears on Codecov.

You may not need a token for public repos. For private repos, create one on Codecov and store it as a GitHub secret.

Part 5 — Dependency updates with Dependabot

Create .github/dependabot.yml:

```
None
version: 2
updates:
  - package-ecosystem: "pip"
    directory: "/"
    schedule:
      interval: "weekly"
```

Dependabot will open PRs for outdated packages.

Part 6 — Security scanning with CodeQL

Create .github/workflows/codeql-analysis.yml:

```
None
name: CodeQL

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]
```

```
jobs:
  analyze:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Initialize CodeQL
        uses: github/codeql-action/init@v1
        with:
          languages: python

      - name: Perform CodeQL Analysis
        uses: github/codeql-action/analyze@v1
```

If the action warns about deprecated versions, update the @v1 tags to the latest major available in GitHub's docs.

Part 7 — Matrix builds (multiple Python versions)

Update jobs.test in test.yml:

None

```
jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: [3.10, 3.11, 3.12]
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: ${{ matrix.python-version }}
      - run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt
```

```
      - run: pytest --cov=app test_app.py
```

Part 8 — Extend the API (POST, PUT, DELETE)

Implement additional endpoints (e.g., POST /echo) and add a test. Example:

Python

```python
# In app.py
@app.route('/echo', methods=['POST'])
def echo():
    data = request.get_json(force=True)
    return jsonify(data), 201
```

Python

```python
# In test_app.py
from flask import json
import app

def test_echo():
    client = app.app.test_client()
    payload = {"msg": "ping"}
    response = client.post('/echo', json=payload)
    assert response.status_code == 201
    assert response.get_json() == payload
```

Push changes and confirm CI runs your new tests.

Part 9 — (Optional) Deploy to AWS

High level (choose one approach):

- Elastic Beanstalk: Use aws-actions/configure-aws-credentials and a deploy step that calls eb deploy.
- EC2: SSH or use the AWS CLI to copy artifacts and restart a service (e.g., gunicorn).
- ECS/Fargate: Build & push a Docker image to ECR, then update the ECS service.

Add any secrets/credentials via Repo Settings → Secrets and variables → Actions.

Troubleshooting

- Module not found / pytest missing: Ensure pytest, pytest-cov, and flask are in requirements.txt and installed in CI.
- Codecov upload fails: Add CODECOV_TOKEN for private repos; confirm coverage files exist (.coverage, coverage.xml if configured).
- Flake8 errors: Fix style violations or configure ignores with setup.cfg / tox.ini / .flake8.
- CodeQL warnings about versions: Update the action tag to the current major version suggested by GitHub.
- Matrix build failures: Verify your code supports all selected Python versions or adjust the matrix.