Demo: Automating CI/CD Pipelines Using Generative AI and Enhancing Deployment Speed and Quality

---

Steps:

Step 1: Setup a Sample Repository

Objective: Set up a sample repository where the CI/CD pipeline will be applied.

Instructions:

1. Create a new GitHub repository:
   - Go to GitHub and create a new repository for your project.
   - Clone the repository to your local machine:
     ```bash
     git clone https://github.com/your-username/your-repository.git
     cd your-repository
     ```

2. Create a sample application (a simple Python app, for example):
   - In your local repository, create a file `app.py`:
     ```python
     def greet(name):
         return f"Hello, {name}"

     if __name__ == "__main__":
         print(greet("World"))
     ```

```

3. Create a requirements file:

- Create a file `requirements.txt` to specify dependencies:

```txt
pytest
```

4. Commit and push the changes to GitHub:

```bash
git add .
git commit -m "Initial commit with sample app"
git push origin main
```

Step 2: Generate an Automated CI/CD Pipeline Using GitHub Actions

Objective: Use GitHub Copilot to automatically generate a CI/CD pipeline configuration file using GitHub Actions.

Instructions:

1. Create the GitHub Actions workflow file:

- In your repository, create a new directory `.github/workflows/` and add a file called `ci.yml`:

```bash
mkdir -p .github/workflows
touch .github/workflows/ci.yml
```
```

2. Write a comment to prompt GitHub Copilot to generate the pipeline:

- Open `ci.yml` in your editor (e.g., Visual Studio Code) and add the following comment at the top:

```yaml
GitHub Actions CI/CD pipeline for building and testing a Python app
```

3. Let GitHub Copilot generate the pipeline:

- GitHub Copilot will suggest something like this:

```yaml
name: CI

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Set up Python
      uses: actions/setup-python@v2
      with:
        python-version: '3.x'
```

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

      - name: Run tests
        run: |
          pytest
```

4. Explanation of the Generated Pipeline:
   - The pipeline is triggered on `push` or `pull_request` events to the `main` branch.
   - It runs on the latest Ubuntu environment and performs the following steps:
     - Checkout the code: This retrieves the code from the repository.
     - Set up Python: This configures the correct version of Python.
     - Install dependencies: It installs the necessary dependencies listed in `requirements.txt`.
     - Run tests: This runs the `pytest` tests to ensure that the code is working properly.

Step 3: Integrating Testing Tools and Deployment Targets

Objective: Add deployment targets to the pipeline and integrate testing tools.

Instructions:

1. Write a prompt for GitHub Copilot to add a deployment step:
   - In the same `ci.yml` file, after the `Run tests` step, write the following comment to deploy the app:

```yaml
```

Add a step to deploy the app to Heroku (or any other deployment target)

```

2. Let GitHub Copilot generate the deployment step:

- GitHub Copilot will generate a deployment step. For example, deploying to Heroku:

```yaml

- name: Deploy to Heroku

  run: |

    git push https://heroku:your-api-key@git.heroku.com/your-app-name.git main

```

3. Explanation:

- The code generated by Copilot pushes the code to a Heroku application. You can replace this with any deployment target (e.g., AWS, Azure, GCP).

4. Commit and push the changes:

```bash

git add .github/workflows/ci.yml

git commit -m "Add CI/CD pipeline with deployment"

git push origin main

```

Step 4: Automating Build Pipelines with Generative AI

Objective: Use GitHub Copilot to automate Dockerfile generation and CI/CD pipelines.

Instructions:

1. Write a prompt for Dockerfile generation:
```

- In your repository, create a `Dockerfile`:

  ```bash
  touch Dockerfile
  ```

- In the `Dockerfile`, write a prompt to generate a Dockerfile:

  ```dockerfile
  Generate a Dockerfile to build a Python app
  ```

2. Let GitHub Copilot generate the Dockerfile:

   - GitHub Copilot will generate something like:

     ```dockerfile
     FROM python:3.9-slim

     WORKDIR /app

     COPY requirements.txt requirements.txt
     RUN pip install -r requirements.txt

     COPY . .

     CMD ["python", "app.py"]
     ```

3. Build and test the Docker image:

   ```bash
   docker build -t my-python-app .
   docker run -it my-python-app
   ```

4. Update the GitHub Actions pipeline to include Docker build:

   - In `ci.yml`, add a step to build the Docker image:

     ```yaml
     - name: Build Docker image
       run: |
         docker build -t my-python-app .
     ```