Deploying a simple Python Flask application using **Docker**, **DockerHub**, **GitHub Actions**, and **AWS EC2 (Ubuntu)** involves several steps. I will need to search for the best practices and specific commands for each part of this process.

Here are the search queries to gather the necessary information:

This is a common and robust CI/CD workflow. I'll provide a step-by-step guide to set up the infrastructure, the application, and the automated pipeline.

The overall flow is:

Local Development → GitHub → GitHub Actions (Build Image → Push to DockerHub) → AWS EC2 (Ubuntu) (SSH connection pulls and runs the new image).

## 1. Local Setup and Dockerization

You'll need three core files in your local project directory.

### A. app.py (Simple Flask Application)

This is your main application file.

Python

from flask import Flask  
  
app = Flask(\_\_name\_\_)  
  
@app.route('/')  
def hello():  
 return "Hello from the Dockerized Flask App on AWS EC2!"  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # It's crucial for Flask to listen on 0.0.0.0 for it to be accessible  
 # outside the container (e.g., from the EC2 host).  
 app.run(host='0.0.0.0', port=5000)

### B. requirements.txt (Dependencies)

List your Python dependencies.

Plaintext

flask

### C. Dockerfile (Container Definition)

This defines how to build your Docker image.

Dockerfile

# Use a slim Python base image  
FROM python:3.11-slim  
  
# Set the working directory in the container  
WORKDIR /app  
  
# Copy requirements file and install dependencies  
COPY requirements.txt .  
RUN pip install --no-cache-dir -r requirements.txt  
  
# Copy the rest of the application code  
COPY . .  
  
# Expose the port the app runs on (Flask's default is 5000)  
EXPOSE 5000  
  
# Command to run the application  
CMD ["python", "app.py"]

## 2. AWS EC2 Setup

Set up your target deployment environment.

### A. Launch EC2 Instance

1. Go to the **AWS Management Console** and launch a new **EC2 Instance**.
2. Choose an **Ubuntu Server** AMI (e.g., Ubuntu Server 22.04 LTS).
3. Select a free-tier eligible instance type (e.g., t2.micro).
4. **Key pair:** Create a new key pair (.pem file) or use an existing one. **Save this file securely; you will need it for the GitHub Action to SSH.**
5. **Security Group:** Configure a security group with the following inbound rules:
   * **SSH (Port 22):** My IP (recommended) or 0.0.0.0/0 (if you need to access from anywhere).
   * **Custom TCP (Port 5000):** 0.0.0.0/0 (This is the port your Flask app will run on and needs to be accessible).

### B. Install Docker on EC2

SSH into your new Ubuntu EC2 instance using your key pair and public IP, then run these commands:

Bash

# Update and install Docker  
sudo apt update  
sudo apt install -y docker.io  
  
# Start and enable Docker  
sudo systemctl start docker  
sudo systemctl enable docker  
  
# Add the 'ubuntu' user to the docker group so you don't need 'sudo'  
# You'll need to log out and log back in for this change to take effect.  
sudo usermod -aG docker ubuntu

## 3. GitHub Secrets Configuration

GitHub Actions needs secure access to push to DockerHub and deploy to your EC2. Store these as **Repository Secrets** in your GitHub repository under **Settings > Secrets and variables > Actions**.

| Secret Name | Value | Purpose |
| --- | --- | --- |
| DOCKER\_USERNAME | Your DockerHub Username | For logging into DockerHub. |
| DOCKER\_PASSWORD | Your DockerHub **Personal Access Token** (or password) | For pushing the image. |
| EC2\_HOST | Your EC2 Public IP or Public DNS | The address of your EC2 machine. |
| EC2\_USERNAME | ubuntu | The default user on Ubuntu EC2. |
| SSH\_PRIVATE\_KEY | **Contents** of your EC2 .pem key file | For the GitHub Action to SSH into EC2. |

⚠️ **Important:** For SSH\_PRIVATE\_KEY, copy the *entire content* of your .pem file, starting with -----BEGIN RSA PRIVATE KEY----- and ending with -----END RSA PRIVATE KEY-----.

## 4. GitHub Actions Workflow

Create a file at .github/workflows/main.yml in your repository. This workflow will run on every push to the main branch.

YAML

name: CI/CD Pipeline to EC2  
  
on:  
 push:  
 branches:  
 - main  
 # Allows you to run the workflow manually from the GitHub Actions tab  
 workflow\_dispatch:  
  
jobs:  
 build-and-deploy:  
 runs-on: ubuntu-latest  
   
 steps:  
 - name: 1. Checkout repository code  
 uses: actions/checkout@v4  
  
 - name: 2. Login to Docker Hub  
 uses: docker/login-action@v3  
 with:  
 username: ${{ secrets.DOCKER\_USERNAME }}  
 password: ${{ secrets.DOCKER\_PASSWORD }}  
  
 - name: 3. Set up image tag  
 id: set\_tag  
 run: echo "IMAGE\_TAG=${{ secrets.DOCKER\_USERNAME }}/flask-app:latest" >> $GITHUB\_ENV  
   
 - name: 4. Build and push Docker image to Docker Hub  
 uses: docker/build-push-action@v5  
 with:  
 context: .  
 push: true  
 tags: ${{ env.IMAGE\_TAG }}  
  
 - name: 5. Deploy to AWS EC2 via SSH  
 uses: appleboy/ssh-action@v1.0.3  
 with:  
 host: ${{ secrets.EC2\_HOST }}  
 username: ${{ secrets.EC2\_USERNAME }}  
 key: ${{ secrets.SSH\_PRIVATE\_KEY }}  
 # The following commands run on your EC2 instance  
 script: |  
 echo "Starting deployment on EC2..."  
 # 1. Log in to Docker Hub on EC2 (optional, but good practice for rate limits)  
 echo "${{ secrets.DOCKER\_PASSWORD }}" | docker login -u ${{ secrets.DOCKER\_USERNAME }} --password-stdin  
   
 # 2. Pull the latest image  
 docker pull ${{ env.IMAGE\_TAG }}  
   
 # 3. Stop and remove the old container (if running)  
 docker stop flask-container || true  
 docker rm flask-container || true  
   
 # 4. Run the new container, mapping EC2's port 80 to container's port 5000  
 # We're using port 80 on EC2 for standard HTTP access.  
 docker run -d \  
 -p 80:5000 \  
 --name flask-container \  
 --restart always \  
 ${{ env.IMAGE\_TAG }}  
   
 echo "Deployment complete! Application should be accessible on port 80."

## 5. Final Step: Verification

1. Commit and push your app.py, requirements.txt, Dockerfile, and the new .github/workflows/main.yml to your GitHub repository's main branch.
2. Go to the **Actions** tab in GitHub to watch the workflow run.
3. Once the job succeeds, open your AWS EC2 Security Group and ensure **Inbound Rule** for **HTTP (Port 80)** is open to 0.0.0.0/0.
4. Open a web browser and navigate to your **EC2 Public IP Address**.

You should see the message: **"Hello from the Dockerized Flask App on AWS EC2!"**

Any future push to your main branch will automatically re-run this process, update the image on DockerHub, and deploy the new version to your EC2 instance with zero manual intervention.