Lab 6: Automate Docker Image Deployment with GitHub Actions

Lab overview

In this lab, you will explore automating Docker image deployment using GitHub Actions with advanced CI/CD features. You'll create deployment pipelines, set up GitHub environments with protection rules, implement manual approval workflows, and trigger deployments automatically or manually. By the end of this lab, you'll have a complete automated deployment system with staging and production environments.

In this lab, you will:

- Automate Docker image deployment from CI pipeline
- Trigger deployments on main branch or manually
- Use GitHub environments and approval workflows
- Set up staging and production deployment environments
- Implement environment protection rules and secrets
- Handle rollback and deployment status monitoring

Estimated completion time

60 minutes

Task 1: Creating and containerizing the web application

In this task, you will create a simple web application that will be containerized and deployed automatically.

1. To create a folder named **Lab6** on the Desktop, right-click on the Desktop, click **New**, then select **Folder**. Name the folder **Lab6**. Open in VSCode.
2. Create the main application file.
   2.1. In VSCode, click on the **Explorer** icon in the left sidebar (or press **Ctrl+Shift+E**).
   2.2. Click the **New File** icon next to Lab6 folder name.
   2.3. Name the file **app.py**.
   2.4. Click on **app.py** to open it in the editor.
   2.5. Add the following code.

```
"""Simple Flask Web Application for Deployment Demo"""


from flask import Flask, render_template_string, jsonify
import os
import datetime


app = Flask(__name__)


# HTML template
HTML_TEMPLATE = """
<!DOCTYPE html>
<html>
<head>
    <title>Deployment Demo App</title>
    <style>
```

```
        body { font-family: Arial, sans-serif; margin: 40px;
background: #f4f4f4; }

        .container { background: white; padding: 30px; border-radius:
8px; box-shadow: 0 2px 10px rgba(0,0,0,0.1); }

        .header { color: #333; text-align: center; }

        .info { background: #e8f4fd; padding: 15px; border-radius:
5px; margin: 20px 0; }

        .status { background: #d4edda; padding: 10px; border-radius:
5px; color: #155724; }

        .version { font-size: 0.9em; color: #666; }

    </style>

</head>

<body>

    <div class="container">

        <h1 class="header">🚀 Deployment Demo Application</h1>

        <div class="info">

            <h3>Environment: {{ env }}</h3>

            <p><strong>Version:</strong> {{ version }}</p>

            <p><strong>Deployment Time:</strong> {{ deploy_time }}</p>

            <p><strong>Container ID:</strong> {{ container_id }}</p>

        </div>

        <div class="status">

            ☑ Application is running successfully!

        </div>

        <div class="version">

            <p>This application demonstrates automated Docker
deployment with GitHub Actions.</p>

        </div>
```

```
    </div>
</body>
</html>
"""

@app.route('/')
def home():
    """Home page with deployment information."""
    return render_template_string(HTML_TEMPLATE,
        env=os.getenv('ENVIRONMENT', 'development'),
        version=os.getenv('APP_VERSION', '1.0.0'),
        deploy_time=os.getenv('DEPLOY_TIME',
datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')),
        container_id=os.getenv('HOSTNAME', 'local')[:12]
    )

@app.route('/health')
def health():
    """Health check endpoint."""
    return jsonify({
        'status': 'healthy',
        'environment': os.getenv('ENVIRONMENT', 'development'),
        'version': os.getenv('APP_VERSION', '1.0.0'),
        'timestamp': datetime.datetime.now().isoformat()
    })

@app.route('/version')
```

```
def version():
    """Version information endpoint."""
    return jsonify({
        'app_version': os.getenv('APP_VERSION', '1.0.0'),
        'environment': os.getenv('ENVIRONMENT', 'development'),
        'build_number': os.getenv('BUILD_NUMBER', 'local'),
        'git_commit': os.getenv('GIT_COMMIT', 'unknown')
    })


if __name__ == '__main__':
    port = int(os.getenv('PORT', 5000))
    app.run(host='0.0.0.0', port=port, debug=False)
```

2.6. Save the file by pressing Ctrl+S.

3. Create requirements file.
   3.1. Click the New File icon.
   3.2. Name the file requirements.txt.
   3.3. Add the following content.

```
Flask==3.0.0
gunicorn==21.2.0
```

3.4. Save the file by pressing Ctrl+S.

4. Create Dockerfile for containerization.
   4.1. Click the New File icon.
   4.2. Name the file Dockerfile.
   4.3. Add the following content.

```
# Use Python slim image
FROM python:3.11-slim


# Set working directory
```

```
WORKDIR /app

# Copy requirements first for better caching
COPY requirements.txt .

# Install dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code
COPY . .

# Create non-root user
RUN adduser --disabled-password --gecos '' appuser && \
    chown -R appuser:appuser /app
USER appuser

# Expose port
EXPOSE 5000

# Set environment variables
ENV PYTHONPATH=/app
ENV FLASK_APP=app.py

# Run with gunicorn
CMD ["gunicorn", "--bind", "0.0.0.0:5000", "--workers", "2",
"app:app"]
```

4.4. Save the file by pressing **Ctrl+S**.

5. Create Docker Compose for local testing.

5.1. Click the **New File** icon.

5.2. Name the file **docker-compose.yml**.

5.3. Add the following content.

```
version: '3.8'


services:
  app:
    build: .
    ports:
      - "8080:5000"
    environment:
      - ENVIRONMENT=local
      - APP_VERSION=1.0.0
      - DEPLOY_TIME=2025-01-01 12:00:00
    restart: unless-stopped
```

5.4. Save the file by pressing **Ctrl+S**.

6. Verify your file structure. Your VSCode Explorer should now show:

```
Lab6/
├── app.py
├── requirements.txt
├── Dockerfile
└── docker-compose.yml
```

Task 2: Testing locally and push code to GitHub/DockerHub

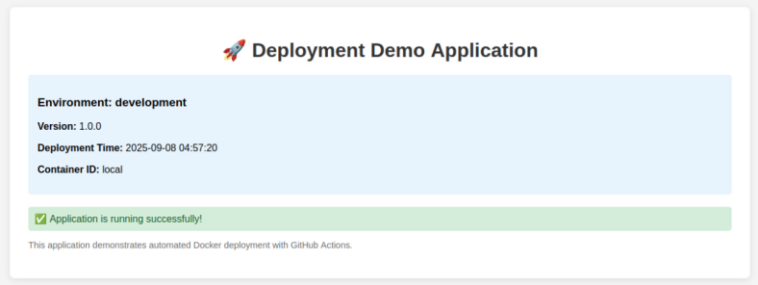
In this task, you will create a GitHub repository and configure environments for staging and production deployments.

1. Test the application locally first.
    1.1. Open VSCode terminal (Terminal > New Terminal).
    1.2. Test the Flask app directly.

```
# Install dependencies locally
pip3 install -r requirements.txt

# Run the application
python3 app.py
```

    1.3. Open the browser to http://localhost:5000 to verify it works.



    1.4. From the terminal. stop the app with Ctrl+C.
2. Test with Docker.

```
# First, verify Docker is working
docker --version
docker ps

# Build the Docker image
docker build -t deployment-demo:latest .
```

```
# Run the container
docker run -d -p 8080:5000 --name demo-app deployment-demo:latest

# Test the application
curl http://localhost:8080/health

# View in browser at http://localhost:8080
# You should see the deployment demo page

# Stop and remove container
docker stop demo-app
docker rm demo-app
```

Expected results:

- ✅ docker build completes successfully
- ✅ Container runs without errors
- ✅ Health endpoint returns JSON response
- ✅ Web page displays deployment information

3. Initialize Git repository.
   3.1. In VSCode terminal, run the following code.

```
git init
git add .
git commit -m "Initial commit: Flask deployment demo app"
```

4. Create GitHub repository.
   4.1. Open your web browser and go to **https://github.com**.
   4.2. Log in to your GitHub account.
   4.3. Click the **+** icon in the top right corner.
   4.4. Select **New repository**.
   4.5. Name the repository **flask-deployment-demo**.
   4.6. Leave it as **Public**.
   4.7. Do NOT initialize with README.
   4.8. Click **Create repository**.
5. Connect local repository to GitHub.
   5.1. Copy the commands from GitHub and run in terminal (replace YOUR_GITHUB_USERNAME).

```
git branch -M main
```

```
git remote add origin https://github.com/YOUR_GITHUB_USERNAME/flask-deployment-demo.git
```

```
git push -u origin main
```

6. Set up a DockerHub account and repository. Create DockerHub account (if needed).
   6.1. Go to **https://hub.docker.com**.
   6.2. Click **Sign Up** if you don't have an account.
   6.3. Choose a **username** (remember this – you'll need it later).
   6.4. Verify your email.

   Find your actual DockerHub username:

   – After logging in, your **username** appears in the top-right corner
   – Example: If you see **"Welcome, user123"**, your **username** is **use123**
   – **Write down your exact username - you'll need it for the next steps**

   Create the DockerHub repository:

   – After logging in to DockerHub, look for **"Create Repository"** button
     - **Option A:** Blue button on dashboard
     - **Option B:** Click **"Repositories"** tab, then "Create Repository"
     - **Option C:** Click "+" icon in top navigation
     - **Option D:** Go directly to https://hub.docker.com/repository/create
   – **Repository name:** flask-deployment-demo

- **Visibility:** Public (free)
- **Description:** "Flask deployment demo for Lab 6"
- Click "Create"

8. Test DockerHub push with your actual username:

```
# First, find your DockerHub username
docker login
# Note the username shown in "Authenticating with existing credentials... [Username: YOUR_USERNAME]"

# Build and tag with YOUR actual DockerHub username
# Replace YOUR_DOCKERHUB_USERNAME with your actual username
docker build -t YOUR_DOCKERHUB_USERNAME/flask-deployment-demo:test .

# Example: If your username is waqar8593
# docker build -t waqar8593/flask-deployment-demo:test .

# Push to verify it works
docker push YOUR_DOCKERHUB_USERNAME/flask-deployment-demo:test

# If successful, clean up
docker rmi YOUR_DOCKERHUB_USERNAME/flask-deployment-demo:test
```

Task 3: Set Up Environments and Secrets in GitHub

1. Create Staging Environment:

- Go to your GitHub repository in the browser
- Click **Settings** tab
- In the left sidebar, click **Environments**
- Click **New environment**
- Name it **staging**
- Click **Configure environment**

Staging Environment Configuration:

- Leave **Required reviewers** empty (no approval needed for staging)
- Under **Environment secrets**, click **Add secret**
- Add these secrets:
  - **Name:** DOCKER_REGISTRY **Value:** docker.io
  - **Name:** DEPLOY_URL **Value:** https://staging-demo.example.com
- Click **Save protection rules**

2. Create Production Environment:

- Click **New environment** again
- Name it **production**
- Click **Configure environment**

Production Environment Configuration:

- Check **Required reviewers**
- Add your GitHub username as a reviewer
- Set **Wait timer** to 1 minutes
- Under **Environment secrets**, click **Add secret**
- Add these secrets:
  - **Name:** DOCKER_REGISTRY **Value:** docker.io
  - **Name:** DEPLOY_URL **Value:** https://demo.example.com
- Click **Save protection rules**

3. Create DockerHub Access Token:

- Go to **https://hub.docker.com** and login
- Click your username → **Account Settings**
- Click **Security** in left sidebar
  - If you can't find Security, try**: https://hub.docker.com/settings/security**
  - Alternative: Use your DockerHub password as the token (less secure)
- Click **New Access Token**
- **Name: GitHub Actions Lab6**
- **Permissions:** Read, Write, Delete
- Click **Generate** and copy the token immediately

4. Set up Github Repository Secrets:

- In repository **Settings**, click **Secrets and variables → Actions**

Add Repository Secrets:

- Click **New repository secret** and add:
  - **Name:** DOCKERHUB_USERNAME **Value:** Your actual DockerHub username (e.g., user123)
  - **Name:** DOCKERHUB_TOKEN **Value:** Your DockerHub access token (or password)
- These will be available to all environments

Task 4: Creating Automated Deployment Workflow

In this task, you will create GitHub Actions workflows for automated deployment with environment-specific configurations.

1. Create GitHub Actions workflow directory:
   - In VSCode Explorer, right-click in the Lab6 folder
   - Select **New Folder**
   - Name it **.github**
   - Right-click on **.github** folder
   - Select **New Folder**
   - Name it **workflows**

2. Create the main deployment workflow:
   - Right-click on **workflows** folder
   - Select **New File**
   - Name it **deploy.yml**
   - Add the following comprehensive workflow:

```
name: Build and Deploy

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]
  workflow_dispatch:
    inputs:
      environment:
        description: 'Environment to deploy to'
        required: true
        default: 'staging'
        type: choice
        options:
```

```
        - staging
        - production
    force_deploy:
      description: 'Force deployment even if tests fail'
      required: false
      default: false
      type: boolean

env:
  IMAGE_NAME: ${{ github.repository }}
  REGISTRY: docker.io

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

    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.11'

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

    - name: Test application
      run: |
        python -c "import app; print('App imports successfully')"
```

```
      python -c "from app import app; client = app.test_client(); resp = client.get('/health');
print(f'Health check: {resp.status_code}')"

  build:
    needs: test
    runs-on: ubuntu-latest
    outputs:
      image-tag: ${{ steps.meta.outputs.tags }}
      image-digest: ${{ steps.build.outputs.digest }}
    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Set up Docker Buildx
      uses: docker/setup-buildx-action@v3

    - name: Login to DockerHub
      if: github.event_name != 'pull_request'
      uses: docker/login-action@v3
      with:
        registry: ${{ env.REGISTRY }}
        username: ${{ secrets.DOCKERHUB_USERNAME }}
        password: ${{ secrets.DOCKERHUB_TOKEN }}

    - name: Extract metadata
      id: meta
      uses: docker/metadata-action@v5
      with:
        images: ${{ env.REGISTRY }}/${{ secrets.DOCKERHUB_USERNAME }}/flask-deployment-
demo
        tags: |
          type=ref,event=branch
          type=ref,event=pr
          type=sha,prefix={{branch}}-
```

```
        type=raw,value=latest,enable={{is_default_branch}}

    - name: Build and push Docker image
      id: build
      uses: docker/build-push-action@v5
      with:
        context: .
        platforms: linux/amd64,linux/arm64
        push: ${{ github.event_name != 'pull_request' }}
        tags: ${{ steps.meta.outputs.tags }}
        labels: ${{ steps.meta.outputs.labels }}
        cache-from: type=gha
        cache-to: type=gha,mode=max
        build-args: |
          BUILD_NUMBER=${{ github.run_number }}
          GIT_COMMIT=${{ github.sha }}

  deploy-staging:
    if: github.ref == 'refs/heads/main' || github.event_name == 'workflow_dispatch'
    needs: build
    runs-on: ubuntu-latest
    environment:
      name: staging
      url: ${{ vars.DEPLOY_URL }}
    steps:
    - name: Deploy to Staging
      id: deploy
      run: |
        echo "🚀 Deploying to staging environment..."
        echo "Image: ${{ needs.build.outputs.image-tag }}"
        echo "Registry: ${{ vars.DOCKER_REGISTRY }}"
        echo "Simulating deployment to staging server..."
        sleep 5
```

```
      echo "☑ Deployment to staging completed!"

    - name: Run staging tests
      run: |
        echo "🚀 Running staging environment tests..."
        echo "☑ All staging tests passed!"

    - name: Notify staging deployment
      run: |
        echo "📢 Staging deployment successful!"
        echo "Environment: staging"
        echo "Version: ${{ github.sha }}"

  deploy-production:
    if: (github.ref == 'refs/heads/main' && github.event_name == 'push') ||
        (github.event_name == 'workflow_dispatch' && github.event.inputs.environment ==
'production')
    needs: [build, deploy-staging]
    runs-on: ubuntu-latest
    environment:
      name: production
      url: ${{ vars.DEPLOY_URL }}
    steps:
    - name: Deploy to Production
      id: deploy
      run: |
        echo "🚀 Deploying to production environment..."
        echo "Image: ${{ needs.build.outputs.image-tag }}"
        echo "Registry: ${{ vars.DOCKER_REGISTRY }}"
        echo "Simulating deployment to production server..."
        sleep 10
        echo "☑ Deployment to production completed!"
```

```
  - name: Run production smoke tests
    run: |
      echo "🚀 Running production smoke tests..."
      echo "☑ All production tests passed!"

  - name: Notify production deployment
    run: |
      echo "📢 Production deployment successful!"
      echo "Environment: production"
      echo "Version: ${{ github.sha }}"
```

- Save the file by pressing Ctrl+S

3. Create a rollback workflow:
   - Right-click on **workflows** folder
   - Select **New File**
   - Name it **rollback.yml**
   - Add the following rollback workflow:

```
name: Rollback Deployment

on:
 workflow_dispatch:
  inputs:
    environment:
      description: 'Environment to rollback'
      required: true
      type: choice
      options:
        - staging
        - production
    previous_version:
      description: 'Previous version/tag to rollback to'
      required: true
      type: string
```

```
jobs:
  rollback:
    runs-on: ubuntu-latest
    environment: ${{ github.event.inputs.environment }}
    steps:
    - name: Rollback deployment
      run: |
        echo "🔄 Rolling back ${{ github.event.inputs.environment }} to version ${{ github.event.inputs.previous_version }}"
        echo "Simulating rollback process..."
        sleep 5
        echo "☑ Rollback completed successfully!"

    - name: Verify rollback
      run: |
        echo "🔍 Verifying rollback..."
        echo "☑ Rollback verification passed!"

    - name: Notify rollback
      run: |
        echo "📢 Rollback completed!"
        echo "Environment: ${{ github.event.inputs.environment }}"
        echo "Rolled back to: ${{ github.event.inputs.previous_version }}"
```

- Save the file by pressing Ctrl+S

4. Create a status badge and README:
   - Click the **New File** icon in Lab6 folder
   - Name it **README.md**
   - Add the following content (replace YOUR_GITHUB_USERNAME and YOUR_DOCKERHUB_USERNAME):

```
# Flask Deployment Demo
```

```
![Build and Deploy](https://github.com/YOUR_GITHUB_USERNAME/flask-deployment-demo/workflows/Build%20and%20Deploy/badge.svg)
![Docker](https://img.shields.io/badge/docker-%230db7ed.svg?style=flat&logo=docker&logoColor=white)
![Flask](https://img.shields.io/badge/flask-%23000.svg?style=flat&logo=flask&logoColor=white)

A demonstration of automated Docker deployment with GitHub Actions, environments, and approval workflows.

## Features

- 🚀 **Automated Deployment**: Push to main triggers staging deployment
- 🔒 **Environment Protection**: Production requires manual approval
- 🐳 **Docker**: Containerized application with multi-platform builds
- 🔄 **Rollback**: Manual rollback workflow for quick recovery
- 📊 **Monitoring**: Health checks and deployment status
- 🌍 **Multi-Environment**: Staging and Production environments

## Environments

- **Staging**: Auto-deployed on main branch push
- **Production**: Requires manual approval after staging success

## Docker Repository

- **DockerHub**: [YOUR_DOCKERHUB_USERNAME/flask-deployment-demo](https://hub.docker.com/r/YOUR_DOCKERHUB_USERNAME/flask-deployment-demo)

## Manual Deployment

To deploy manually:
1. Go to **Actions** tab
2. Select **Build and Deploy** workflow
```

3. Click **Run workflow**
4. Choose environment and options
5. Click **Run workflow**

Rollback

To rollback a deployment:
1. Go to **Actions** tab
2. Select **Rollback Deployment** workflow
3. Click **Run workflow**
4. Choose environment and previous version
5. Click **Run workflow**

Local Development

```bash
# Run locally
pip install -r requirements.txt
python app.py

# Run with Docker
docker build -t flask-demo .
docker run -p 8080:5000 flask-demo
```

- Replace `YOUR_GITHUB_USERNAME` with your actual GitHub username
- Replace `YOUR_DOCKERHUB_USERNAME` with your actual DockerHub username - Save the file

Deployment Flow

Push to main → Tests → Build Image → Deploy Staging → (Approval) → Deploy Production

Task 5: Testing Automated Deployment

In this task, you will test the complete deployment pipeline including manual approvals and rollbacks.

1. Test complete build and run cycle:

```
# Navigate to Lab6 directory
cd ~/Desktop/Lab6

# Build with YOUR actual DockerHub username
# Replace YOUR_DOCKERHUB_USERNAME with your actual username
docker build -t YOUR_DOCKERHUB_USERNAME/flask-deployment-demo:final-test .

# Run the container
docker run -d -p 8080:5000 --name final-test YOUR_DOCKERHUB_USERNAME/flask-deployment-demo:final-test

# Test the endpoints
curl http://localhost:8080/health
curl http://localhost:8080/version

# Check in browser: http://localhost:8080
# Should show: "Deployment Demo Application"

# Clean up
docker stop final-test && docker rm final-test
docker rmi YOUR_DOCKERHUB_USERNAME/flask-deployment-demo:final-test
```

✅ All commands must work without sudo before proceeding!

2. Commit and push the workflows:

```
git add .
git commit -m "Add automated deployment workflows with environments"
git push origin main
```

3. **Monitor the automatic deployment:**
   - Go to your GitHub repository in the browser
   - Click the **Actions** tab
   - You should see a **Build and Deploy** workflow running
   - Click on the workflow run to see detailed progress

4. **Observe the deployment flow:**

   Build Phase:
   - ✅ Tests run and pass
   - ✅ Docker image builds and pushes to DockerHub

Staging Deployment:
   - ✅ Automatically deploys to staging environment
   - ✅ Runs staging tests
   - ✅ Shows deployment URL

Production Deployment:
   - ⏳ Waits for manual approval (yellow circle)
   - 👤 Requires your approval to proceed

5. **Test manual approval for production:**
   - In the workflow run, you'll see **production** environment waiting
   - Click **Review deployments**
   - Select **production** environment
   - Add a comment like "Approved for production deployment"
   - Click **Approve and deploy**
   - Watch the production deployment proceed

6. **Test manual workflow dispatch:**
   - In the **Actions** tab, click **Build and Deploy**
   - Click **Run workflow** button (top right)
   - You'll see options:
     - **Environment:** Choose staging or production
     - **Force deploy:** Check or uncheck
   - Select staging and click **Run workflow**
   - Watch the targeted deployment run

7. **Test the rollback workflow:**
   - Go to **Actions** tab
   - Click **Rollback Deployment**
   - Click **Run workflow**
   - Fill in the inputs:
     - **Environment:** Choose staging
     - **Previous version:** Enter v1.0.0
   - Click **Run workflow**
   - Watch the rollback simulation

8. Test deployment failure and recovery:
   - Make a change that will break the tests:
   - Edit app.py and introduce an error:
   - Add this line at the top level of app.py after the imports:

```
from flask import Flask, render_template_string, jsonify
import os
import datetime

# Add this line to test failure
test_failure = undefined_variable_will_cause_import_error

app = Flask(__name__)
```

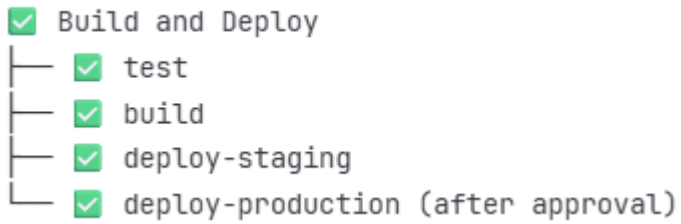
   - Commit and push:

```
git add app.py
git commit -m "Test deployment failure"
git push origin main
```

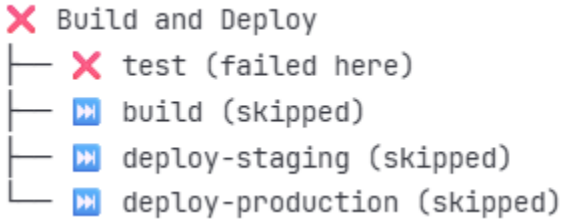
   - Watch the workflow fail at the test stage
   - Fix the code and push again to see recovery
9. View deployment history:
   - Go to Actions → Deployments
   - Click on staging or production
   - See all deployments with status and details

10. Understanding workflow outputs:

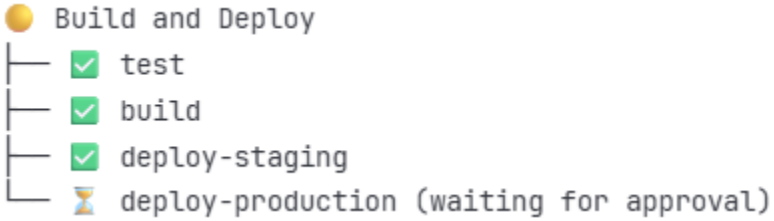
Successful Deployment:



Failed Deployment:



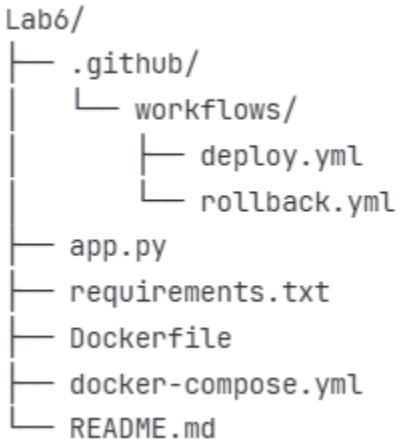
Manual Approval Pending:



11. Verify DockerHub Integration:

- Go to your DockerHub repository: https://hub.docker.com/r/YOUR_DOCKERHUB_USERNAME/flask-deployment-demo
- You should see the pushed images with different tags (main, latest, etc.)
- Each successful workflow run creates a new image tag

12. Final file structure:



Lab review

What is the primary benefit of using GitHub Environments with required reviewers?

A. To speed up deployment processes
B. To prevent unauthorized deployments to critical environments
C. To reduce Docker image size
D. To automatically fix deployment failures

Correct Answer: B. To prevent unauthorized deployments to critical environments

STOP

You have successfully completed this lab.