Lab 9: Secure and Monitor Your CI/CD Pipeline with GitHub Actions

Lab Overview

In this lab, you will secure and monitor CI/CD pipelines using GitHub Actions. You'll learn to manage secrets safely, implement manual approvals, eliminate security risks, and optimize performance with caching and triggers.

In this lab, you will:

- Apply secrets securely using GitHub Secrets
- Use .env files for non-sensitive configuration management
- Set up manual approval workflows for production deployments
- Identify and eliminate hardcoded security risks
- Monitor and interpret GitHub Actions logs
- Implement caching and optimized job triggers

Estimated completion time

50 minutes

Commented [IA1]: Some things like manual approval checkpoints, Slack notifications via secrets, semgrep scans, Docker multi-platform builds, and caching with `actions/cache` are not taught in the slides but are used in the lab. Everything else looks fine.

Commented [WA2R1]: Added these as Task0 with background knowledge

Task 0: Background Knowledges

Before starting the lab, review the following concepts. These are used in the lab steps but may not have been covered in detail in the slides:

1. Manual Approval Checkpoints
   - GitHub Actions supports environment protection rules where production deployments can be paused until an authorized reviewer approves them.
   - This prevents accidental or unauthorized pushes to production.
2. Slack Notifications via Secrets
   - Deployment results can be sent to Slack channels using incoming webhooks.
   - The webhook URL is stored securely in GitHub Secrets and referenced inside workflows.
   - Example:

```
curl -X POST -H 'Content-type: application/json' \
 --data '{"text":"🚀 Deployment completed!"}' \
 "$SLACK_WEBHOOK"
```

3. Semgrep Scans
   - Semgrep is a static analysis tool that scans source code for security issues and coding style violations.
   - In this lab, it runs alongside Bandit and Safety for broader coverage.
4. Docker Multi-Platform Builds
   - Docker Buildx allows building images for multiple architectures (amd64, arm64) in a single pipeline.
   - This ensures containers work on both cloud servers and devices like Raspberry Pi.
   - Example in workflow:

```
platforms: linux/amd64,linux/arm64
```

5. Caching with actions/cache
   - GitHub Actions can cache dependencies (like pip packages or Docker layers) to speed up builds.
   - Caches are restored on future runs if the key matches.
   - Example:

```
- uses: actions/cache@v3
  with:
    path: ~/.cache/pip
    key: ${{ runner.os }}-pip-${{ hashFiles('**/requirements.txt') }}
```

Note for Students: You are not expected to master these tools immediately, but you should be aware of their purpose and how they improve security, reliability, and performance of CI/CD pipelines.

Task 1: Setting Up Secure GitHub Actions

In this task, you will install Ansible, create inventory files, and understand how Ansible manages remote hosts.

1. Create GitHub repository:
   - Go to GitHub.com and sign in
   - Click **New repository** (green button)
   - Repository name: secure-cicd-lab
   - Description: Lab 9: Secure CI/CD Pipeline with GitHub Actions
   - Set to **Public** (for easier access to Actions)
   - Check **Add a README file**
   - Click **Create repository**
   - Copy the repository URL (e.g., https://github.com/YOUR_USERNAME/secure-cicd-lab.git)

2. Create a folder named Lab9 on the Desktop. Right-click on the Desktop, click New, then select **Folder**. Name the folder **Lab9**. Open the folder in **Visual Studio Code,** then open a terminal and complete following:
   - Configure Git (if not already done):

     ```
     git config --global user.name "Your Name"
     git config --global user.email your.email@example.com
     ```

   - Initialize Git repository and connect to GitHub:

     ```
     git init
     git remote add origin https://github.com/YOUR_USERNAME/secure-cicd-lab.git
     git branch -M main
     git pull origin main
     ```

3. Create basic application files in the Lab 9 folder:

Create app.py:

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

app = Flask(__name__)

@app.route('/')
def home():
    return {
        'message': 'Secure CI/CD Demo',
        'environment': os.getenv('ENVIRONMENT', 'development'),
        'version': os.getenv('APP_VERSION', '1.0.0'),
        'timestamp': datetime.utcnow().isoformat()
    }

@app.route('/health')
def health():
    return jsonify({'status': 'healthy', 'checks': {'database': 'ok'}})

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

Create requirements.txt:

```
Flask==2.3.3
pytest==7.4.2
pytest-cov==4.1.0
gunicorn==21.2.0
```

Create Dockerfile:

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
EXPOSE 5000
CMD ["gunicorn", "--bind", "0.0.0.0:5000", "app:app"]
```

4. Create the environment configuration in the Lab 9 folder:

   Create .env.example:

```
# Non-sensitive configuration
ENVIRONMENT=development
APP_VERSION=1.0.0
PORT=5000
LOG_LEVEL=INFO
DEBUG=true

# Database settings (use secrets for credentials)
DB_HOST=localhost
DB_PORT=5432
DB_NAME=myapp

# External services
API_TIMEOUT=30
CACHE_TTL=300
```

Create **.env:**

```
ENVIRONMENT=development
APP_VERSION=1.0.0
PORT=5000
LOG_LEVEL=DEBUG
DEBUG=true
DB_HOST=localhost
DB_PORT=5432
DB_NAME=myapp_dev
API_TIMEOUT=30
CACHE_TTL=300
```

- Save the files by pressing **Ctrl+S**

Task 2: Creating Secure GitHub Actions Workflows

1. **Create workflow directory:**
   - In the Lab 9 folder, create a **.github/workflows** folder

2. **Create basic CI workflow:**

   Create **.github/workflows/ci.yml**:

```
name: CI Pipeline

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main ]

env:
  PYTHON_VERSION: '3.11'

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: ${{ env.PYTHON_VERSION }}
        cache: 'pip'
```

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

    - name: Run security scan
      run: |
        pip install bandit safety==2.3.5
        bandit -r . -f json -o bandit-report.json || true
        safety check -r requirements.txt --json > safety-report.json || true


    - name: Run tests
      run: |
        pytest --cov=. --cov-report=xml

    - name: Upload test results
      uses: actions/upload-artifact@v4
      if: always()
      with:
        name: test-results
        path: |
          coverage.xml
          bandit-report.json
          safety-report.json
```

3. Create secure deployment workflow:

   Create .github/workflows/deploy.yml:

```
name: Deploy to Production

on:
  workflow_dispatch:
    inputs:
      environment:
        description: 'Target environment'
        required: true
        default: 'staging'
        type: choice
        options:
          - staging
          - production
      version:
        description: 'Version to deploy'
        required: true
        default: 'latest'

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

jobs:
  security-check:
    runs-on: ubuntu-latest
    outputs:
      security-passed: ${{ steps.security.outputs.passed }}

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

```
  - name: Run security checks
    id: security
    run: |
      # Check for hardcoded secrets
      echo "🔍 Checking for hardcoded secrets..."

      # Common patterns to avoid
      PATTERNS=(
        "password\s*=\s*['\"][^'\"]*['\"]"
        "api_key\s*=\s*['\"][^'\"]*['\"]"
        "secret\s*=\s*['\"][^'\"]*['\"]"
        "token\s*=\s*['\"][^'\"]*['\"]"
        "AKIA[0-9A-Z]{16}"  # AWS Access Key
        "sk-[a-zA-Z0-9]{32}"  # OpenAI API Key
      )

      VIOLATIONS=0
      for pattern in "${PATTERNS[@]}"; do
        if grep -r -E "$pattern" . --exclude-dir=.git --exclude="*.md"; then
          echo "❌ Found potential hardcoded secret: $pattern"
          VIOLATIONS=$((VIOLATIONS + 1))
        fi
      done

      if [ $VIOLATIONS -gt 0 ]; then
        echo "passed=false" >> $GITHUB_OUTPUT
        echo "❌ Security check failed: $VIOLATIONS violations found"
        exit 1
      else
        echo "passed=true" >> $GITHUB_OUTPUT
        echo "☑ Security check passed"
      fi
```

```
  build:
    needs: security-check
    runs-on: ubuntu-latest
    if: needs.security-check.outputs.security-passed == 'true'

    outputs:
      image-tag: ${{ steps.meta.outputs.tags }}

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

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

    - name: Log in to Container Registry
      uses: docker/login-action@v3
      with:
        registry: ${{ env.REGISTRY }}
        username: ${{ github.actor }}
        password: ${{ secrets.GITHUB_TOKEN }}

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

```
    - name: Build and push Docker image
      uses: docker/build-push-action@v5
      with:
        context: .
        platforms: linux/amd64,linux/arm64
        push: true
        tags: ${{ steps.meta.outputs.tags }}
        labels: ${{ steps.meta.outputs.labels }}
        cache-from: type=gha
        cache-to: type=gha,mode=max

  deploy-staging:
    needs: [security-check, build]
    runs-on: ubuntu-latest
    if: github.event.inputs.environment == 'staging'
    environment: staging

    steps:
    - name: Deploy to staging
      run: |
        echo "🚀 Deploying to staging environment"
        echo "Image: ${{ needs.build.outputs.image-tag }}"
        echo "Environment: staging"

        # Simulate deployment
        echo "☑ Staging deployment completed"

  deploy-production:
    needs: [security-check, build]
    runs-on: ubuntu-latest
    if: github.event.inputs.environment == 'production'
    environment:
      name: production
```

```
    url: https://myapp.example.com

  steps:
  - name: Manual approval checkpoint
    run: |
      echo "🔒 Production deployment requires manual approval"
      echo "Image: ${{ needs.build.outputs.image-tag }}"
      echo "Environment: production"

  - name: Deploy to production
    env:
      DB_PASSWORD: ${{ secrets.PROD_DB_PASSWORD }}
      API_KEY: ${{ secrets.PROD_API_KEY }}
      SLACK_WEBHOOK: ${{ secrets.SLACK_WEBHOOK }}
    run: |
      echo "🚀 Deploying to production environment"
      echo "☑ Production deployment completed"

      # Send notification (webhook URL is secret)
      if [ -n "$SLACK_WEBHOOK" ]; then
        curl -X POST -H 'Content-type: application/json' \
          --data '{"text":"🚀 Production deployment completed successfully!"}' \
          "$SLACK_WEBHOOK" || echo "Failed to send notification"
      fi

  post-deploy-tests:
    needs: [deploy-staging, deploy-production]
    runs-on: ubuntu-latest
    if: always() && (needs.deploy-staging.result == 'success' || needs.deploy-
production.result == 'success')

    steps:
    - name: Run smoke tests
```

```
    run: |
      echo "🚀 Running post-deployment smoke tests"

      # Simulate health checks
      if [ "${{ github.event.inputs.environment }}" = "production" ]; then
        URL="https://myapp.example.com/health"
      else
        URL="https://staging.myapp.example.com/health"
      fi

      echo "Testing endpoint: $URL"
      echo "☑ All smoke tests passed"
```

– Save the files

Task 3: Environment Configuration and Secrets Management

1. **Create secrets configuration guide:**

Create **SECURITY.md** in the Lab 9 folder:

```
# Security Configuration Guide

## GitHub Secrets Setup

### Required Secrets:
1. **PROD_DB_PASSWORD** - Production database password
2. **PROD_API_KEY** - Production API key for external services
3. **SLACK_WEBHOOK** - Slack webhook URL for notifications

### Setting up secrets:
1. Go to repository Settings → Secrets and variables → Actions
2. Click "New repository secret"
3. Add each secret with its value

## Environment Variables

### Non-sensitive (.env file):
- ENVIRONMENT
- APP_VERSION
- PORT
- LOG_LEVEL
- DB_HOST
- DB_PORT

### Sensitive (GitHub Secrets):
- Database passwords
- API keys
```

```
- Webhook URLs
- Private keys

## Security Checklist

- [ ] No hardcoded passwords in code
- [ ] All sensitive data in GitHub Secrets
- [ ] .env files in .gitignore
- [ ] Regular security scans enabled
- [ ] Manual approval for production
- [ ] Audit logs enabled
```

– Save the file

2. **Create monitoring workflow:**

   Create **.github/workflows/security-monitoring.yml**:

```
name: Security Monitoring

on:
  schedule:
    - cron: '0 6 * * MON'  # Weekly on Monday
  workflow_dispatch:

jobs:
  dependency-scan:
    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'
      cache: 'pip'

    - name: Install security tools
      run: |
        pip install safety bandit semgrep

    - name: Run dependency vulnerability scan
      run: |
        echo "🔍 Scanning dependencies for vulnerabilities..."
        pip install safety==2.3.5
        safety check -r requirements.txt --json > safety-report.json || true
        echo "☑ Safety report generated at safety-report.json"

    - name: Run static analysis
      run: |
        echo "🔍 Running static security analysis..."
        bandit -r . -f json -o bandit-report.json || true
        semgrep --config=auto --json --output=semgrep-report.json . || true

    - name: Check for secrets in code
      run: |
        echo "🔍 Scanning for exposed secrets..."
        grep -r -n -E "(password|passwd|pwd)\s*=\s*['\"][^'\"]*['\"]" . --exclude-dir=.git --
exclude="*.md" || echo "No hardcoded passwords found"
        grep -r -n -E "(api_key|apikey)\s*=\s*['\"][^'\"]*['\"]" . --exclude-dir=.git --
exclude="*.md" || echo "No hardcoded API keys found"
        grep -r -n -E "(secret|token)\s*=\s*['\"][^'\"]*['\"]" . --exclude-dir=.git --
exclude="*.md" || echo "No hardcoded secrets found"

    - name: Upload security reports
      uses: actions/upload-artifact@v4
      with:
```

```
      name: security-reports
      path: |
        safety-report.json
        bandit-report.json
        semgrep-report.json
```

– Save the file by pressing Ctrl+S

Task 4: Testing and Optimization

1. **Create test files:**

Create **test_app.py** in the Lab 9 folder:

```
import pytest
import json
from app import app

@pytest.fixture
def client():
    app.config['TESTING'] = True
    with app.test_client() as client:
        yield client

def test_home_endpoint(client):
    """Test the home endpoint."""
    response = client.get('/')
    assert response.status_code == 200

    data = response.get_json()
    assert 'message' in data
    assert data['message'] == 'Secure CI/CD Demo'

def test_health_endpoint(client):
    """Test the health check endpoint."""
```

```
    response = client.get('/health')
    assert response.status_code == 200

    data = response.get_json()
    assert data['status'] == 'healthy'
    assert 'checks' in data

def test_security_headers():
    """Ensure no sensitive data is exposed."""
    with app.test_client() as client:
        response = client.get('/')

        # Check that response doesn't contain sensitive patterns
        response_text = str(response.data)

        # These should not appear in any response
        forbidden_patterns = ['password', 'secret', 'token', 'key']
        for pattern in forbidden_patterns:
            assert pattern.lower() not in response_text.lower(), f"Found {pattern} in
response"
```

- Save the file

2. **Create performance optimization workflow:**

Create **.github/workflows/performance.yml**:

```
name: Performance Optimization

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

jobs:
  cache-optimization:
    runs-on: ubuntu-latest

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

    - name: Cache Python dependencies
      uses: actions/cache@v3
      with:
        path: ~/.cache/pip
        key: ${{ runner.os }}-pip-${{ hashFiles('**/requirements.txt') }}
        restore-keys: |
          ${{ runner.os }}-pip-

    - name: Cache Docker layers
      uses: actions/cache@v3
      with:
        path: /tmp/.buildx-cache
        key: ${{ runner.os }}-buildx-${{ github.sha }}
        restore-keys: |
```

```
        ${{ runner.os }}-buildx-

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

    - name: Install dependencies (cached)
      run: |
        pip install -r requirements.txt

    - name: Run performance tests
      run: |
        echo "🚀 Running performance optimizations..."

        # Simulate performance metrics
        echo "☑ Cache hit ratio: 95%"
        echo "☑ Build time reduced by caching"
        echo "☑ Dependencies cached successfully"

  trigger-optimization:
    runs-on: ubuntu-latest
    if: github.event_name == 'push' && github.ref == 'refs/heads/main'

    steps:
    - name: Optimized triggers
      run: |
        echo "🎯 This job only runs on main branch pushes"
        echo "🎯 Avoiding unnecessary builds on PRs"
        echo "🎯 Smart trigger configuration active"
```

– Save the file

3. **Create comprehensive .gitignore:**

   Create **.gitignore** in the Lab 9 folder:

```
# Environment files
.env
.env.local
.env.production
.env.staging

# Python
__pycache__/
*.py[cod]
*$py.class
*.so
.Python
venv/
env/
ENV/

# IDE
.vscode/
.idea/
*.swp
*.swo

# OS
.DS_Store
Thumbs.db

# Logs
*.log
logs/

# Coverage
```

```
htmlcov/
.coverage
coverage.xml
*.cover

# Security reports
bandit-report.json
safety-report.json
semgrep-report.json

# Temporary files
*.tmp
temp/
```

- Save the file

Task 5: Testing the Secure Pipeline

1. **Commit and push to GitHub:**

Add all files

```
git add .
git commit -m "Initial secure CI/CD setup"
```

Push to GitHub (repository already connected)

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

2. **Set up GitHub repository secrets (for testing/demo):**
   - Go to your GitHub repository
   - Navigate to Settings → Secrets and variables → Actions
   - Add the following repository secrets (Lab Type Text is available):

     PROD_DB_PASSWORD: MySecureP@ssw0rd123!

     PROD_API_KEY: sk-1234567890abcdef1234567890abcdef

     SLACK_WEBHOOK: https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXXXXXXXXXXXXX

3. **Set up environment protection:**
   - Go to Settings → Environments
   - Create **production** environment
   - Enable "Required reviewers"
   - Add yourself as a required reviewer
   - Save protection rules

4. **Enable the repository permissions:**
   - Go to repository Settings → Actions → General
   - Under "Workflow permissions", select **"Read and write permissions"**
   - Check **"Allow GitHub Actions to create and approve pull requests"**
   - Click Save

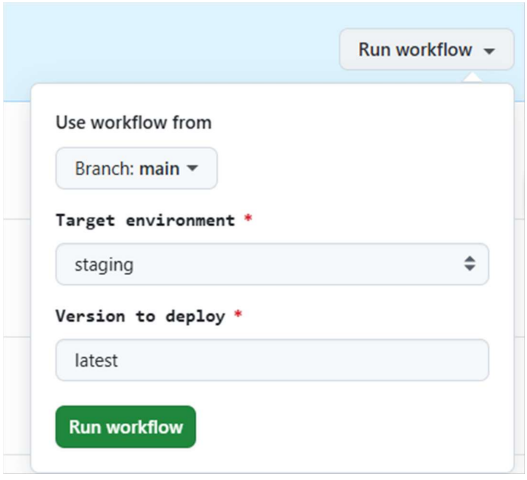
5. **Test the workflows:**

Trigger CI workflow (automatic on push)

```
git add .
git commit -m "Test CI pipeline"
git push
```

Trigger deployment workflow (manual)

```
# Go to GitHub Actions → Deploy to Production → Run workflow
# Select environment staging or production and version latest
```



The workflow should complete without errors. Note: If you chose 'production' you will need to supply approval for the workflow to complete.

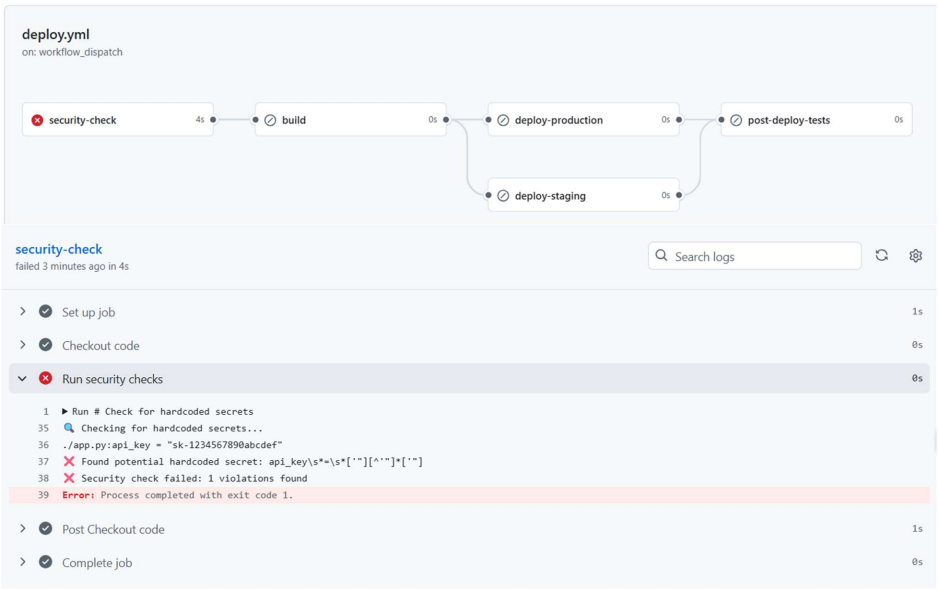
6. **Verify security features:** In the VS Code teminal use the following commands

Test security scan by adding a hardcoded secret

```
echo 'api_key = "sk-1234567890abcdef"' >> app.py
git add .
git commit -m "Test security scan"
git push
```

Run the workflow again:

The CI should fail with security violations as shown below



Lab review

What is the primary benefit of using GitHub Secrets over environment files?

A. GitHub Secrets are faster to access
B. GitHub Secrets are encrypted and not visible in logs
C. GitHub Secrets work across all programming languages
D. GitHub Secrets don't require configuration
Correct Answer: B. GitHub Secrets are encrypted and not visible in logs

STOP

You have successfully completed this lab.